高等学校工科类

研究生教学用书

智能优化方法

Intelligent Optimization Methods

汪定伟　王俊伟　王洪峰　张瑞友　郭　哲　编著



高等教育出版社

ISBN 978-7-04-020886-3



定价 27.70 元

■ 学科类别：信息技术

高等学校工科类
研究生教学用书

智能优化方法

Intelligent Optimization Methods

汪定伟　王俊伟　王洪峰　张瑞友　郭　哲　编著

高等教育出版社

内容提要

本教材主要介绍近年来产生发展的多种智能优化算法。包括为人熟知的遗传算法、禁忌搜索算法、模拟退火算法和蚁群优化算法；近年来已成为研究热点的粒子群优化算法；还有尚待普及的捕食搜索算法和动态环境下的进化计算。书中讨论这些算法的产生和发展、算法的基本思想和理论、基本构成、计算步骤和主要的变形以及数值例子和实际应用。为了方便读者学习，各章之后还附有精选的习题、思考题及相关的参考文献。

本教材是为“智能优化方法”这门研究生课程编写的，可作为系统工程、管理工程、计算机、自动化、人工智能以及其他应用优化算法专业的研究生及高年级的本科生教材，也可供相关专业的研究人员和工程技术人员参考。

图书在版编目（CIP）数据

智能优化方法/汪定伟等编著. —北京：高等教育出版社，2007.4

ISBN 978-7-04-020886-3

Ⅰ.智… Ⅱ.汪… Ⅲ.最优化算法-研究 Ⅳ.O242.23

中国版本图书馆 CIP 数据核字（2007）第 024543 号

策划编辑	刘　英	**责任编辑**	欧阳舟	**封面设计**	李卫青
责任绘图	吴文信	**版式设计**	史新薇	**责任校对**	殷　然
责任印制	宋克学				

出版发行	高等教育出版社	**购书热线**	010-58581118
社　　址	北京市西城区德外大街4号	**免费咨询**	800-810-0598
邮政编码	100011	**网　　址**	http://www.hep.edu.cn
总　　机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
			http://www.landraco.com.cn
经　　销	蓝色畅想图书发行有限公司	**畅想教育**	http://www.widedu.com
印　　刷	高等教育出版社印刷厂		
开　　本	787×960　1/16	**版　　次**	2007年4月第1版
印　　张	20	**印　　次**	2007年4月第1次印刷
字　　数	330 000	**定　　价**	27.70元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

物料号　20886-00

前　言

智能优化方法是一个近年来发展起来的非常活跃的研究领域。系统工程、自动化、计算机、管理工程、采矿、机械等许多专业的学者和学生都在广泛地采用智能优化方法。比如，遗传算法、禁忌搜索算法、模拟退火算法、蚁群算法和粒子群优化算法等在国民经济的各个行业中都获得了广泛的应用。

目前，国内出版的一些智能优化方法相关著作主要集中在遗传、禁忌搜索和模拟退火等发展较早的算法方面，而近年来备受各界人士推崇的蚁群算法和粒子群优化算法以及捕食搜索算法、动态进化算法介绍很少。另外，国内出版的有关智能优化方法书籍大多是学术性较强的专著，并不适于一般学生学习使用，读者群也相对较小。

编者自1996年起就开始在东北大学讲授“智能优化方法”这门研究生课程，并在研究中大量应用这些算法来解决各种实际中的优化问题，积累了很多学习、应用和传授这些算法的经验。我们的目的是为希望了解、学习这个领域的学生和学者提供一本通俗易懂、由浅入深的教科书，而不是一本对某个专题深入讨论、艰深的学术专著。本教材把智能优化的各种主要方法都囊括其中，包括：遗传算法、禁忌搜索算法、模拟退火算法、蚁群算法、粒子群优化算法、捕食搜索算法和动态进化算法。所以本教材是一本对智能优化方法进行全面介绍的入门书。

本教材每一章中，主要介绍算法的产生、算法的基本思想和理论、算法的基本构成、计算步骤、主要的变形算法、几个数值举例和应用，而不去深入地讨论算法的理论。各章之后附有精选的参考文献，有深入研究兴趣的读者，还可以根据参考文献的索引去查阅其他专门的文献和书籍。为了方便学生学习，各章后都附有习题与思考题，通过课后练习可以加深学生对课程内容的理解。

本教材共分为9章，各章节的内容安排如下：第1章介绍智能优化方法的产生与发展；第2章介绍伪随机数的产生；第3章～第7章是本书的主干，分别介绍遗传算法、禁忌搜索算法、模拟退火算法、蚁群算法和粒子群优化算法；第8章介绍近年来新出现的捕食搜索算法；第9章讨论动态进化算法；最后在结束语中介绍了作者对未来优化方法发展趋势的看法。

本教材是根据汪定伟教授的课程讲义编写的。由于原讲义主要讲述遗传算法、禁忌搜索算法和模拟退火算法，其他算法仅仅只有一个简介，而且这些算法很多内容都需要更新，因此各撰写人为之付出了很多心血和劳动。各章节的撰写分工如下：

汪定伟：第1章，第2章和结束语

王俊伟：第3章，第7章和第8章

王洪峰：第5章和第9章

张瑞友：第4章

郭哲：第6章

最后由汪定伟校阅、统调、定稿。由于智能优化方法发展很快，相关名词和术语亦更新较快，故难以统一。限于编者的水平，书中的错误之处在所难免，恳请广大读者指正。

本教材的出版得到了东北大学研究生优秀教材基金的资助，并且得到了国家自然科学基金重点项目和创新团队项目（70431003，60521003）以及教育部流程工业综合自动化重点实验室的经费资助，在此表示感谢。本教材在编写和出版中还得到了高等教育出版社研究生教育与学术分社刘英编辑和其他同志的支持和帮助，在此一并表示衷心的感谢。

编　者

2006年10月于沈阳

目　　录

第1章　智能优化方法的产生与发展

本章首先介绍最优化的重要意义，然后从分析传统优化方法的基本步骤及其局限性入手，讨论实际中对新的优化方法的需求，介绍智能优化方法的产生、发展和主要特点。最后简单地介绍近年来最优化发展的一些新动向。

1.1　最优化的重要意义

人类一切活动的实质不外乎是“认识世界，建设世界”。认识世界靠的是建立模型，简称建模；建设世界靠的是优化决策，所以“建模与优化”可以说无所不在，它们始终贯穿在一切人类活动的过程之中。

从概念模型、结构模型，到数学模型以及计算机仿真模型和实物模型，是模型的不同阶段。从某种意义上说，人类的一切知识不外乎是人类对某个领域的现象和过程认识的模型。只是由于不同领域问题的模型化的难易程度不同，其模型处在不同的阶段。比如，数学、力学、微观经济学等，其知识基本上是用数学模型来表达的；而哲学、社会学、心理学等，由于许多因素难以定量化，其模型大多还处在概念模型阶段。

认识世界的目的是为了建设世界，同样建模的目的就是为了优化。建设世界首先必须认识世界，同样一切优化都离不开模型。比如，建设一个水电站首先要认识河流的水文规律，而只有综合考虑淹没损失、水坝造价和发电效益，选择最优的建设方案，才能确保水电站建设的成功。

最优化离不开模型，所以最优化方法的发展正是随着模型描述方法的发展而发展起来的。代数学中解析函数的发展，产生了极值理论，这是最早的无约束的函数优化方法。而拉格朗日乘子法则是最早的约束优化方法。第二次世界大战时期，英国为了最有效地利用有限的战争资源，成立了作战研究小组，取得了良好的效果。战后，作战研究的优化思想被运用到运输管理、生产管理和一些经济学问题中，于是形成了以线性规划、博弈论等为主干的运筹学。运筹学的英文名正是“作战研究（Operation Research)”，其精髓就是要在用约束条件表述的限制下，

实现用目标函数表述的某个目标的最优化。线性规划、非线性规划、动态规划、博弈论、排队论、存储论等，这些运筹学的模型使最优化方法的发展达到了极致，从而开启了最优化的辉煌时代。

除了在军事领域里的成功运用，最优化在国际经济的各个领域里都获得了广泛的运用。运输计划、工厂选址、设备布置、生产计划、作业调度、商品定价、材料切割、广告策略、路径选择、工作指派……各种各样的典型问题都在应用最优化方法。钢铁、采矿、运输、制造业等，各行各业都在运用最优化。

对个人来说，家庭理财、职业选择、人生计划、作息安排，生活的方方面面都可以运用最优化方法。可以说，最优化是人类智慧的精华，会不会最优化是个人聪明才智的表征。最优化水平的高低直接反映了一个人智力和受教育水平的高低。

这本书讲述的就是最新且最实用的优化方法。

1.2　传统优化方法的基本步骤及其局限性

1. 传统优化方法的基本步骤

传统的优化方法主要指：线性规划的单纯形法，非线性规划的基于梯度的各类迭代算法。这类算法的基本步骤包括如下3个步骤，如图1.1所示。

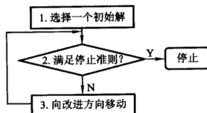


图1.1　传统优化方法的基本步骤

第1步：选择一个初始解

传统的优化方法总是从选择初始解开始，一般说来，这个初始解还必须是可行解。比如线性规划的单纯形法，首先要用大M法，或二阶段法来找到一个基础可行解。对于无约束的非线性函数优化问题，初始解一般可以任选，但是对带约束的非线性规划问题，通常也必须选择可行解作为初始解。

第2步：判断停止准则是否满足

这一步是对现行解检验是否满足停止准则。停止准则通常就是最优

性条件。比如，对于线性规划的单纯形法，若检验数向量

$$\pi = C_B^{\mathrm{T}} B^{-1} N - C_N^{\mathrm{T}} \geqslant 0 \tag{1.1}$$

则满足最优性条件，停机；否则转入下一步迭代。

这里，B、N 分别是约束矩阵中基础变量和非基础变量对应的部分；C_B和 C_N是价格向量中基础变量和非基础变量对应的部分。

对于无约束的非线性函数优化问题，检验梯度函数 $\nabla f(x^k)=0$是否成立。

对于非线性规划问题，则必须检验 Kuhn - Turker 条件

$$\nabla f(x^k) - \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{h}(x^k) - \boldsymbol{\pi}^{\mathrm{T}} \boldsymbol{g}(x^k) = 0 \tag{1.2}$$

是否成立。其中，$\boldsymbol{h}(x)$ 和 $\boldsymbol{g}(x)$ 分别是等式约束和不等式约束函数向量。

第 3 步：向改进方向移动

当最优性条件不能满足时，就必须向改进解的方向移动。比如对于线性规划的单纯形法，即做转轴变换，旋出一个基础变量，旋入一个非基础变量。对于非线性规划的最速下降法、共轭梯度法、变尺度法等，则向负梯度方向、负共轭梯度方向或负的修正的共轭梯度方向移动。即

$$x^{k+1} = x^k - \alpha \nabla f(x^k) \tag{1.3}$$

这里，α 是移动步长，通常用一维搜索的方法来确定。$\nabla f(x^k)$ 表示当前解 x^k的梯度、共轭梯度或修正的共轭梯度方向。

2. 传统优化方法的局限性

传统优化方法的这种计算构架给它带来了一些难以克服的局限性。这些局限性主要表现在以下几个方面。

（1）单点运算方式大大限制了计算效率的提高

传统优化方法是从一个初始解出发，每次迭代中也只对一个点进行计算，这种方法很难发挥出现代计算机高速计算的性能。特别是高性能的多 CPU 的计算机和现代并行计算模式在传统优化方法中很难应用，这样就限制了算法计算速度和求解大规模问题的能力。

（2）向改进方向移动限制了跳出局部最优的能力

传统的优化方法要求每一步迭代都向改进方向移动，即每一步都要求能够降低目标函数值，这样算法就不可能具有“爬山”能力。一旦算法进入某个局部的低谷，就只能局限在这个低谷区域内，不可能搜索该区域之外的其他区域。这样，算法就失去了宝贵的全局搜索能力。

（3）停止条件只是局部最优性的条件

传统最优化方法的梯度为零或 Kuhn - Turker 条件只是最优解的必要条件，并不是充分必要条件。因此这个条件即使从理论上看也不是充

分的，即满足停止条件的解也不能保证就是最优解。只有当解的可行域是凸集，目标函数是凸函数时，即满足所谓的“双凸”条件时，才能保证获得的解是全局最优解。这种“双凸”条件对于大多数实际问题往往很难满足，这就大大限制了传统优化方法的应用范围。

（4）对目标函数和约束函数的要求限制了算法的应用范围

传统的优化方法通常要求目标函数和约束函数是连续可微的解析函数，有的算法甚至要求这些函数是高阶可微的，比如牛顿法。实际中，这样的条件往往很难满足。比如，价格可能存在批量折扣，生产能力可能有跳跃性变化，机器开、停有起动费用，这些因素都可能造成目标函数只是分段连续的。这样，传统优化方法对目标函数和约束函数的严格要求使其应用范围大打折扣。

任何一种新方法在其产生的初期往往是“方法定向（Method Oriented）”的，即它只能解决满足该方法适用条件的问题。要想运用这种方法就必须简化或改变原来的问题，使之能够满足该方法的适用条件。比如，为了使用线性规划超强的计算能力，实际中往往不得不采用拟线性化或分段线性化的方法把非线性问题转化成线性问题。20 世纪 70 年代末流行过这样一个十分形象的比喻，说最优化方法好像是“只卖一个尺码鞋的鞋店”，脚小的塞棉花，脚大的砍一截。

传统优化方法是初级阶段的优化方法。随着人们对优化方法的要求的提高，它的这种“方法定向”的特征引起了人们的非议和质疑，于是在 20 世纪 70 年代前后，运筹学的发展出现了一个低谷期。这正是传统优化方法局限性的真实写照。

1.3 智能优化方法的产生与发展

针对传统优化方法的不足，人们对最优化提出了一些新的需求。这些需求主要包括以下几个方面。

1. 对目标函数和约束函数表达的要求必须更为宽松

实际问题希望目标函数和约束函数可以不必是解析的，更不必是连续和高阶可微的。目标函数和约束函数中可以含有规则、条件和逻辑关系，甚至只要一段计算机程序可以描述的关系能够输出一个返回值，就可以作为目标函数或约束函数使用。这样，分段连续函数、“IF…THEN…”语句都可以用来表述目标和约束。于是，以规则形式表达的知识和人的经验都可以嵌入到优化模型之中。这样的模型已经不再是传统的数学模型，而是智能模型。

2. 计算的效率比理论上的最优性更重要

传统的优化方法是方法定向的，所以它比较注重理论的最优性。但是实际问题并不介意获得的解是不是理论上最优的，而更加注重的是计算的效率。由于实际问题的复杂性，往往造成问题的规模很大，时效性很高。比如，复杂制造系统的实时调度问题要求优化算法算得快，能解决的问题规模大。这就要求优化方法能够高效快速地找到满意的解，至于是不是最优解反而并不十分重要。

3. 算法随时终止能够随时得到较好的解

传统的优化方法不能保证随时终止时能够获得较好的解，比如非线性规划的外点法，计算中途终止算法连可行解都不能得到。许多实际问题有很高的时效性要求，对于这类问题，虽然计算更长时间可以获得更好的解，但由于急于使用结果往往要求能够随时终止计算，并且在终止时能够获得一个与计算时间代价相当的较好解。

4. 对优化模型中数据的质量要求更加宽松

传统的优化方法是基于精确数学的方法，这类方法对数据的确定性和准确性有严格的要求。实际生活中很多信息具有很高的不确定性，有些只能用随机变量或模糊集合，乃至语言变量来描述。虽然传统的随机规划或模糊优化方法有一定的处理数据不确定性的能力，但这些方法不外乎是用数学期望来替代随机变量，或是将模糊变量清晰化，而且计算的效能很低。实际中迫切希望能够直接对具有不确定性的数据乃至语言变量进行计算的优化方法。

实际生活中对最优化方法性能的需求促进了最优化方法的发展，最优化逐步走出“象牙塔”，面向实际需要，完成了从“方法定向”向“问题定向（Problem Oriented)”的转换。于是新的优化方法不断出现。

1975 年，Holland 提出遗传算法（Genetic Algorithms)。这种优化方法模仿生物种群中优胜劣汰的选择机制，通过种群中优势个体的繁衍进化来实现优化的功能。

1977 年，Glover 提出禁忌搜索（Tabu Search）算法。这种方法将记忆功能引入到最优解的搜索过程中，通过设置禁忌区阻止搜索过程中的重复，从而大大提高了寻优过程的搜索效率。

1983 年，Kirkpatrick 提出模拟退火（Simulated Annealing）算法。这种算法模拟热力学中退火过程能使金属原子达到能量最低状态的机制，通过模拟的降温过程按玻耳兹曼（Boltzmann）方程计算状态间的转移概率来引导搜索，从而使算法具有很好的全局搜索能力。

20 世纪 90 年代初，Dorigo 等提出蚁群优化（Ant Colony Optimiza-

tion）算法。这种算法借鉴蚂蚁群体利用信息素相互传递信息来实现路径优化的机理，通过记忆路径信息素的变化来解决组合优化问题。

1995年，Kennedy和Eberhart提出粒子群优化（Particle Swarm Optimization）算法。这种算法模仿鸟类和鱼类群体觅食迁徙中，个体与群体协调一致的机理，通过群体最优方向、个体最优方法和惯性方向的协调来求解实数优化问题。近年来该方法已经成为新的研究热点。

1999年，Linhares提出捕食搜索（Predatory Search）算法。这种算法模拟猛兽捕食中大范围搜寻和局部蹲守的特点，通过设置全局搜索和局部搜索间变换的阈值来协调两种不同的搜索模式，从而实现了对全局搜索能力和局部搜索能力的兼顾。

此外，近年来，还有模仿食物链中物种相互依存的人工生命算法（Artificial Life Algorithms）；模拟人类社会多种文化间的认同、排斥、交流和改变等特性的文化算法（Cultural Algorithms）等一些各具特点但知名度不够高的智能优化算法提出。

相对传统的优化方法，以上算法有一些共同的特点。

（1）不以达到某个最优性条件或找到理论上的精确最优解为目标，而是更看重计算的速度和效率。

（2）对目标函数和约束函数的要求十分宽松。

（3）算法的基本思想都是来自对某种自然规律的模仿，具有人工智能的特点。

（4）多数算法含有一个多个体的种群，寻优过程实际上就是种群的进化过程。

（5）这些算法的理论工作相对比较薄弱，一般说来都不能保证收敛到最优解。

从这些不同的特点出发，这类算法获得了各种不同的名称。由于算法理论薄弱，它们最早被称为“现代启发式（Modern Heuristics）”或“高级启发式（Advanced Heuristics）”；从其人工智能的特点，还被称为“智能计算（Intelligent Computation）”或“智能优化算法（Intelligent Optimization Algorithms）”；从不以精确解为目标的特点，它们又被归到“软计算（Soft Computing）”方法中；从种群进化的特点看，它们又可以称为“进化计算（Evolutionary Computation）”；从它们模仿自然规律的特点出发，近几年又有人将它们称为“自然计算（Natural Computing）”。当然，这些不同的计算方法名称各自都还有本身的一些新概念，限于本书的中心不在于此，这里不再展开讨论。

从应用这些方法的角度看，以上方法叫什么名称并不重要，重要的

是要掌握它们的特点，知其所长，也知其所短，这样才能对各类问题应用适当的方法。

1.4 怎样学习研究智能优化方法

智能优化方法是一门计算科学。它的理论工作相对比较薄弱，其知识点主要在于介绍各种优化算法的基本思想、计算步骤和计算机实现的技巧。对于大学理工科的学生，建议在学习研究智能优化方法时应该从以下几个方面入手。

1. 应用智能优化算法解决各类问题是重点

智能优化算法对各类复杂的优化问题有很强的适应性，应用这些算法解决一些其他算法难以解决的优化问题就成了研究的重点。由于以上提到的智能优化算法基本上都是“问题依赖（Problem Dependent）”的，即算法的处理细节上会因问题的不同而不同。这样不同的应用问题就需要具体情况具体处理，这就给算法的研究带来了很多要解决的问题。比如，还没有人用新算法解决过的经典的组合优化问题、网络和图论中的优化问题，还有实践中的各种各样的应用问题。近年来杂志上发表的论文基本上都是算法应用类型的。

2. 算法改进有很大的创新空间

智能优化算法大多给出的只是一个基本的计算思想和步骤，因此改进算法步骤以获得更好的计算性能有很大的创新空间。比如，为遗传算法设计新的遗传算子，为模拟退火设计新的冷却策略，为禁忌搜索定义新的邻域搜索，等等，这些都是可以充分发挥读者聪明才智的地方。

3. 多种算法结合的混合算法是一条捷径

由于不同的智能化算法各有特点，怎样将两种乃至两种以上的算法结合起来就一直是不少学者的工作重点。比如，遗传算法和禁忌搜索的混合算法，禁忌搜索和模拟退火的混合算法，等等。当然还有更多的可能的混合方法还没有人实践或者没有成功，这就给后来人留下很多工作的空间。由于不同算法的混合机理是现存的，又容易引起同行们的关注，获得成果的可能性比较大，所以说这是一条成功的捷径。

4. 不提倡刻意去追求理论成果

智能优化算法不是一门理论严谨的学科，而是一门实验学科。它没有什么严格的公理体系，主要是依据计算机计算得到的性能的好坏来判别算法的成功与否。虽然多种算法都有一些理论分析和收敛性证明。但是从严谨数学的角度看，这些理论工作都差强人意，有的甚至经不起仔

细推敲。而且这些理论工作对算法的性能提高并没有多少指导意义，起码没有一个算法按研究的收敛性条件去制定算法停止准则。因此，建议工科的学生不要刻意追求智能优化算法的理论成果，而要将更多的精力投入计算实践中。当然在计算实践中忽然有了理论创新的灵感，也不应该轻易放过。

5. 算法性能的测算是一项要下真功夫的工作

学习研究智能优化算法就要有坐在计算机前反反复复调试程序、计算例题的决心。无论算法的改进、提高还是创新，唯一的评价标准就是大量不同规模例题的试算结果的好坏。虽然创造性不体现在例题测算上，但做算法研究的人大部分时间都用于例题测算。要想在这个领域里获得成功就必须喜欢编程序、喜欢在计算机上工作，并能够从计算性能的意想不到的提高中获得成功的快乐。具有这种潜质的学生是从事智能优化算法研究的最佳人选。

6. 选择测试例题的一般规律

算法性能测试是算法研究的基础工作，那么如何选择测试例题则是算法测试首先要解决的问题。从例题的说服力出发，选择例题的优先顺序为：网上题库中的例题→文献中的例题→随机产生的例题→实际应用问题→自己编的例题。

对于经典的组合优化问题，比如旅行商问题（TSP）、二次指派问题（QAP）等互联网上的经典题库中有很多不同规模的问题，如果你的算法能够取得优于文献中报道的性能指标，那就是一个了不起的成果。对于没有题库的问题，如果有文献计算过，则应该对相同的问题来计算比较。对于新问题，则应该用伪随机数发生器产生不同规模的问题来测试算法的计算性能。再退而求其次就是计算实际问题，这使其他同行将很难判断算法的好坏。最没有说服力的例题就是自己编的例题，虽然有不少国内杂志上的论文就是这样做的，但这种做法实在不值得提倡。

7. 算法性能测算的主要指标

算法性能测算到底测算哪些指标？一般说来，算法性能测试主要包括以下三个方面。

①达优率：即在多次从不同随机种子出发的计算中达到最优解的百分比。当找不到问题的最优解时，可以用计算中获得的最好解替代。由此派生出来的指标还有：解的目标值平均值和最优解的目标值的比、所有解的目标值的标准差等。

②计算速度：在特定的软、硬件环境中计算不同规模问题的计算

时间。

③计算大规模问题的能力：在可接受时间里能够求解的最大问题的规模，比如能够在几小时内求解的 TSP 问题的城市数量。虽然可解问题的规模依赖于计算速度，但同时这个指标还受算法对计算机存储空间需求量的影响。占用存储空间过大的算法，显然就不能求解大规模问题。

8. 创造出新算法是很多人的梦想

学习研究智能优化算法的过程中，很多有创新精神的人都会自然想到能不能创造出新的智能优化算法。但是，创造新算法绝对不是一件容易的事，更不是叫出一个新名词就算成功了。对于一个新算法，要想成功必须达到以下几点。

①有新的思想和新的计算机理。

②至少对某类问题的计算性能优于已有算法。

③能在国际期刊上发表论文并被一些同行引用。

④能有其他作者测试、改进，并应用到其他问题中。

近年来虽然有不少作者提出一些新算法，比如：鱼群算法、雁队算法、群落选址算法等，这种精神值得鼓励，但要真正成为一种被国内外同行认可的算法还需要做大量工作。

问题与思考

1. 试按传统优化算法的三大步骤来分析线性规划的单纯形法的计算步骤。

2. 试按传统优化算法的三大步骤来分析非线性规划的最速下降法的计算步骤。

3. 试举出一个不能用传统优化方法求解的实际问题来具体说明传统优化算法的局限性。

4. 试举出一个实际问题说明问题对优化算法的新的要求，特别是本章没有提到的新的要求。

5. 列举出本章没有提到的具有智能优化特点的其他算法，并说明把它们归为智能优化算法的理由。

参考文献

[1] Dorigo M, Maniezzo V, Colorni A. The ant system: optimization by a colony of cooperating agents [J]. IEEE Trans. Syst. Man Cybernet, 1996, Part B, 26 (1), 29 -41.

[2] Glover F, Tabu search [J]. ORSA Journal on Computing, 1989, 1:

190 – 206.

[3] Hayashi D et al. Distributed optimization by using artificial life [C]. Tras. IEE Japan, 116 – C (5): 584 – 590, 1996.

[4] Holland J H. Adaptation in Natural and Artificial Systems [C]. Ann Arbor: University of Michigan, 1975.

[5] Kennedy J, Eberhart R C. Particle swarm optimization [C]. Proc. IEEE Int. Conf. Neural Networks, Perth, Australia, 11: 1942 – 1948, 1995.

[6] Kirkpatrick S, Gelatt C, Vecchi P. Optimization by simulated annealing [J]. Science, 1983, 220: 671 – 679.

[7] Linhares. Synthesizing a Predatory Search Strategy for VLSI Layouts [C]. IEEE Trans. On Evolutionary Computation, 3 (2): 147 – 152, 1999.

[8] 方述诚，普森普拉 S. 线性优化及扩展——理论与算法 [M]. 汪定伟等译. 北京：科学出版社，1994.

第 2 章　伪随机数的产生

本章首先介绍伪随机数在智能优化算法中的作用，然后介绍最基本的 0 – 1 均匀分布的伪随机数的产生方法，最后介绍用 0 – 1 均匀分布的伪随机数来构成正态分布的伪随机数的方法，以及用逆变法来产生其他已知概率密度函数的伪随机数的方法。

2.1　伪随机数在智能优化方法中的作用

随机现象是自然过程或人工过程中由多种未知因素共同作用产生的一种只可分析其统计规律却不能预测其发生的不确定现象。这种现象表现为一系列没有规则的数值时就成为随机数。在计算机仿真中人们通常需要用到随机数。由于真正的随机数不可获得，于是人们通常用数字计算机按照某种确定的规则，通过迭代递推运算来产生一系列近似随机分布的数列。这样产生的数列虽然不是由真实的随机现象产生的，但具有类似于随机数的统计性质，可以作为随机数来运用，因此将其称为伪随机数（Pseudo Random Number）。产生这种伪随机数的程序就称为伪随机数发生器（Pseudo Random Number Generator，简称 RNG）。

在智能优化算法中，几乎每一种算法都要用到伪随机数。比如：

（1）遗传算法：随机产生初始种群，用旋轮法选择个体，随机选择交叉点，随机选择变异的基因。

（2）禁忌搜索算法：随机选择初始解，选择多阶段禁忌搜索的初始解。

（3）模拟退火算法：随机选择邻域解，按概率作转移决策。

（4）蚁群算法：随机产生初始蚁群，按概率选择路径。

（5）粒子群优化算法：随机产生初始粒子群，移动方向的随机加权组合。

（6）捕食搜索算法：随机选择初始解，在限制区域内的随机搜索。

动态进化计算是这些算法在动态环境下的变形，自然同样要使用伪随机数。此外，在算法性能测试中随机产生不同规模的例题，从不同的随机种子出发来重复计算，也都要用到伪随机数。可以说，伪随机数发生器是智能优化算法中使用频率最高的子程序。

虽然在各种计算机语言中大多都有产生伪随机数的函数或子程序，但是这些通用的子程序有如下一些不足之处：

（1）随机序列的长度和随机数的字长通常不如自编程序来得更好。

（2）在重复计算中，不如自编的程序容易控制。

（3）除了正态分布之外，一般没有产生其他分布函数的伪随机数的子程序。

鉴于以上这些原因，自己按照伪随机数产生的方法来自编伪随机数发生器子程序是十分必要的。同时从学习的观点看，也只有掌握了计算机产生伪随机数的原理和方法，才能在实际中得心应手地使用。

2.2 产生0－1均匀分布伪随机数的乘同余法

0－1均匀分布的伪随机数是最基本也是最简单的伪随机数，它是生成一切其他分布的伪随机数的基础，同时也是智能优化算法中应用最广的伪随机数。

设 X 是0－1均匀分布的随机变量，x 是 X 的一个取值，即一个均匀分布的伪随机数，记为 $x \in U(0,1)$。其中，U 表示均匀分布，0，1表示分布的区间。X 的密度函数为

$$f(x)=\begin{cases}1, & 0 \leqslant x<1 \\ 0, & \text{其他}\end{cases} \tag{2.1}$$

其分布函数为

$$F(x)=\begin{cases}0, & x<0 \\ x, & 0 \leqslant x<1 \\ 1, & x \geqslant 1\end{cases} \tag{2.2}$$

密度函数 $f(x)$ 和分布函数 $F(x)$ 如图2.1所示。

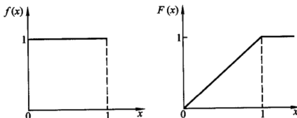


图2.1 0－1均匀分布的密度函数与分布函数

可以容易地推出，$U(0,\ 1)$ 的数学期望 μ 和标准差 σ 分别为

$$\mu=\frac{1}{2},\quad \sigma=\frac{1}{2\sqrt{3}} \tag{2.3}$$

0－1均匀分布的伪随机数通常是由均匀分布的伪随机整数除以数列长度获得的，所以首先要讨论产生均匀随机数的方法。判断一个产生方法好坏的标准主要有以下几点：

（1）产生的数列具有很好的随机性和均匀性。

（2）不出现重复的数列的长度要尽可能长。

（3）产生伪随机数的计算速度要快。

（4）计算程序占用的计算机内存要尽可能少。

产生均匀随机数的方法主要有：

（1）平方取中法：将一个 $2n$ 位的二进制随机数自乘后，取中间的 $2n$ 位作为新的随机数。其公式为

$$B_{k+1}=[B_k,B_k]_{2n} \tag{2.4}$$

其中，B_k表示第 k 次迭代得到的二进制随机数，$[\ \cdot\ ]_{2n}$表示取中间 $2n$ 位数字。

（2）倍积取中法：用一个整数常数 A 来乘一个 n 位的随机数，取中间 n 位作为新的随机数。其公式为

$$S_{k+1}=[A,S_k]_n \tag{2.5}$$

其中，S_k表示第 k 次迭代得到的随机数，$[\ \cdot\ ]_n$表示取中间 n 位数字。

（3）乘同余法：乘同余法是最广泛使用的产生随机数的方法，也是本节讨论的重点。其基本思想是：用一个整数常数 A 来乘一个随机数，将得到的积除以模 M 的余数作为新的随机数。乘同余法用公式表达如下：

设 A 是一个整数常数，M 是一个大的模数，则

$$S_{k+1}=AS_k\mathrm{mod}(M) \tag{2.6}$$

其中，S_k表示第 k 次迭代得到的随机数；$\mathrm{mod}(\ \cdot\ )$ 表示取模运算，即取除以模的余数的运算。

在伪随机数的运用中，总是希望产生不重复的随机数序列的长度越长越好。

根据数论的理论可以证明：位数为 L 的计算机，如果取模数 $M=2^L$，当①$A=8k\pm3$，$A=4k+1$，k 为正整数；②S_0 为奇数时，可以获得最长的随机数序列长度为 2^{L-2}。

例2.1 若 L 为6，则 $M=64$，取 $A=13$，$S_0=1$；那么可以获得一个随机数序列：

{1，13，41，21，17，29，57，37，33，45，9，53，49，61，25，5，1，…}

以上不重复的序列长度为 $2^4=16$，其分布见表2.1。

表2.1　一个随机数序列的分布举例

<10	10~20	20~30	30~40	40~50	50~60	>60
3个	2个	3个	2个	3个	2个	1个

可见以上产生的随机数均匀分布在区间［1，M］内。

（4）混合同余法：混合同余法也称为线性同余法，它能够在同等的计算机上产生比乘同余法更长的均匀分布的随机数序列。其基本思想是：用一个整数常数 A 乘一个随机数再加上一个整数常数 C，将得到的结果除以模 M 的余数作为新的随机数。混合同余法用公式表达如下

$$S_{k+1}=(AS_k+C)\bmod(M) \tag{2.7}$$

显然，当 $C=0$ 时，混合同余法即退化为乘同余法。

根据数论的理论可以证明：位数为 L 的计算机，如果取模数 $M=2^L$，当①$A=4k+1$，k 为正整数；②C 与 M 互为质数时，可以获得最长的随机数序列长度为 2^L。

例2.2　若 L 为5，则 $M=32$，取 $A=13$，$C=5$；仍取 $S_0=1$，那么可以获得一个随机数序列：

{1，18，15，8，13，14，27，4，25，10，7，0，5，6，19，28，17，2，31，24，29，30，11，20，9，26，23，16，21，22，3，12，1，…}

以上随机数序列的不重复的长度为32。

虽然混合同余法比乘同余法要多做一次加法，但在相同计算机上产生的随机数的长度是乘同余法的4倍。在对随机数的长度有较高要求时，应考虑采用混合同余法。

一旦获得随机数 S_i，0-1均匀分布的随机数就可以用除以模获得，公式如下：

$$x_i=S_i/M \tag{2.8}$$

x_i 即为0-1均匀分布的随机数。

［a，b］之间的均匀分布则可以由以下公式获得：

$$y_i=a+(b-a)x_i \tag{2.9}$$

y_i 即为［a，b］之间的均匀分布的随机数，记为 $y_i\in U(a,b)$。

在智能优化算法中经常要产生1和 n 之间的整数 K，这也可以方便地由0-1均匀分布的随机数产生。公式如下：

$$K = 1 + \mathrm{Int}[xn],\ x \in U(0,\ 1) \tag{2.10}$$

其中，Int $[z]$ 为取整运算，即取小于 z 的最大整数。

2.3 产生正态分布伪随机数的方法

正态分布又称高斯分布，是最常见的随机分布，在智能优化方法中也常用到。

若 X 服从参数为 μ 和 σ 的正态分布，即为：$X \sim N(\mu,\ \sigma^2)$ 或 $x \in N(\mu,\ \sigma^2)$。其中，μ 和 σ 分别为 X 的期望值和标准差。当 $\mu = 0$ 和 $\sigma = 1$ 时，称为 0－1 正态分布，即 $N(0,\ 1)$。其密度函数和分布函数分别为

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{2.11}$$

和

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \tag{2.12}$$

0－1 正态分布的密度函数和分布函数如图 2.2 所示。

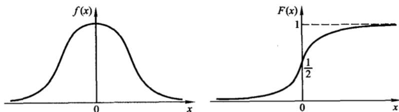


图 2.2 0－1 正态分布的密度函数与分布函数

由于其他正态分布都可以用 $N(0,\ 1)$ 生成，这里只讨论 $N(0,\ 1)$ 的生成方法。

由概率理论知道，若 Y_1，Y_2，…，Y_n 是 n 个独立同分布的随机变量，当 n 足够大时

$$X = Y_1 + Y_2 + \cdots + Y_n \tag{2.13}$$

就近似于一个正态分布。一般说来，当 $n > 6$ 时，X 就是一个正态分布的很好的近似。X 的均值和方差分别为

$$\mu_x = \mu_1 + \mu_2 + \cdots + \mu_n = n\mu \tag{2.14}$$

和

$$\sigma_x^2 = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2 = n\sigma^2 \tag{2.15}$$

于是有 $X \sim N(n\mu,\ n\sigma^2)$ 或 $x \in N(n\mu,\ n\sigma^2)$。为将其化为 $N(0,\ 1)$，令

$$z = \frac{x - \mu_x}{\sigma_x} \tag{2.16}$$

则，$z \in N(0,\ 1)$。

将（2.12）写成随机变量的样本形式，并将式（2.13）和式（2.14）代入式（2.15），得到

$$z = \frac{\sum_{i=1}^{n} y_i - n\mu}{\sqrt{n\sigma^2}} \tag{2.17}$$

可以取 y_i，$i=1,\ 2,\ \cdots,\ n$，为独立的同为 0-1 均匀分布随机变量的样本，则由公式（2.3），得到了确定的 μ 和 σ 的值。代入式（2.16）得到

$$z = \left(\sum_{i=1}^{n} y_i - \frac{n}{2} \right) \Big/ \sqrt{\frac{n}{12}} \tag{2.18}$$

为计算方便，取 $n = 12$，即产生 12 个 0-1 均匀分布的伪随机数 $y_i \in U(0,\ 1)$，$i=1,\ 2,\ \cdots,\ 12$，于是式（2.17）简化为

$$z = \sum_{i=1}^{12} y_i - 6 \tag{2.19}$$

这样就可以方便地得到服从 0-1 正态分布的随机数，$z \in N(0,\ 1)$。

如果需要产生服从一般正态分布 $N(\mu,\ \sigma^2)$ 的随机数 x 时，只要首先按式（2.18）产生 $z \in N(0,\ 1)$，再按公式

$$x = \mu + \sigma z \tag{2.20}$$

变化，就可以获得所需要的伪随机数。

2.4 产生其他分布的伪随机数的逆变法

除了均匀分布和正态分布之外，在智能优化算法中还可能用到其他分布函数的伪随机数，本节讨论产生其他已知分布函数的随机数的逆变法。

设 X 是一个已知分布函数为 $F(x)$ 的随机变量，目的是要产生 X 取值的随机数序列 $\{x_i\}$。设 $Y = F(X)$ 是一个关于 X 的，并以 X 的分布函数 $F(\cdot)$ 为函数随机变量，对于 X 的一个样本值 x，有 $y = F(x)$ 也是 Y 的一个样本值。设 $F^{-1}(\cdot)$为的 $F(\cdot)$ 逆函数，则有

$$X = F^{-1}(Y) \text{和} x = F^{-1}(y) \tag{2.21}$$

那么按照分布函数的定义，Y 的分布函数为

$$
\begin{aligned}
P(Y<y) &= P(F(X)<y) = P(X<F^{-1}(y)) \\
&= P(X<x) = F(x) = y
\end{aligned}
\tag{2.22}
$$

如果 Y 的定义域在 [0, 1] 内，对照式（2.2）表达的 $U(0, 1)$ 的分布函数，可知 Y 服从0－1均匀分布。即 $Y \sim U(0, 1)$，或 $y \in U(0, 1)$。

由以上推导可知，只要产生一个 $y \in U(0, 1)$，按照式（2.20），就可以用分布函数的逆函数 $F^{-1}(\cdot)$ 产生分布函数为 $F(\cdot)$ 的随机数。

有时候给出的是要求产生的随机变量的密度函数 $f(x)$，这时就必须先用

$$
F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t \tag{2.23}
$$

求出分布函数 $F(x)$。

逆变法的计算步骤如下：

步骤1：按式（2.22），由给定的密度函数 $f(x)$ 计算分布函数 $F(x)$，或者已知 $F(x)$。

步骤2：推导 $F(x)$ 的逆函数 $F^{-1}(y)$。

步骤3：产生0－1均匀分布的随机数，$y \in U(0, 1)$。

步骤4：按式 $x = F^{-1}(y)$ 计算得到 X 的一个样本值。

步骤5：重复步骤3至步骤4，即得到一个分布为 $F(x)$ 的随机数序列 $\{x_i\}$。

逆变法为产生其他已知分布的随机数提供了一种十分有效的方法。下面讨论两个逆变法应用的例子。

例2.3 产生负指数分布的随机数 x。

已知负指数分布的密度函数为：$f(x) = \lambda e^{-\lambda x}$，$(x \geqslant 0)$，其分布函数由下式推导：

$$
F(x) = \int_{0}^{x} \lambda e^{-\lambda t}\,\mathrm{d}t = 1 - e^{-\lambda x},\ x \geqslant 0
$$

令 $y = 1 - e^{-\lambda x}$，则求其逆函数得到：$x = -\dfrac{1}{\lambda}\ln(1-y)$。由于当 y 是0－1均匀分布时，$u = 1 - y$ 也是0－1均匀分布的。所以，产生 $u \in U(0, 1)$，按式

$$
x = -\frac{1}{\lambda}\ln u
$$

就可获得负指数分布的随机数 x。

例 2.4 试用逆变法设计分布如图 2.3 所示的密度函数 $f(x)$ 的伪随机数序列的产生方法。

解：由图 2.3 可知，已知密度函数为

$$f(x) = 1 - \frac{x}{2}, \quad 0 \leqslant x < 2$$

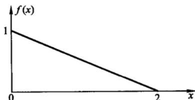


图 2.3 逆变法举例的密度函数

推导得到其分布函数为

$$F(x) = \int_0^x \left(1 - \frac{t}{2}\right) dt = \left[t - \frac{t^2}{4}\right]_0^x = x - \frac{x^2}{4}$$

令 $y = x - \frac{x^2}{4}$，解之得到：$x_1 = 2 + 2\sqrt{1-y}$ 和 $x_2 = 2 - 2\sqrt{1-y}$。由于第一个根不在定义域 [0, 2) 之内，舍去。令 $u = 1 - y$，产生 $u \in U(0, 1)$，按式

$$x = 2 - 2\sqrt{u}$$

即可获得所需要分布的伪随机数。

问题与思考

1. 分别按以下 4 组参数用乘同余法产生 10 个伪随机数，x_1，x_2，…，x_{10}，指出哪组参数设置是正确的，说明不能正确产生伪随机数的参数组的问题出在何处。

(1) $M = 64$，$A = 11$，$x_0 = 1$；(2) $M = 64$，$A = 20$，$x_0 = 1$；(3) $M = 60$，$A = 11$，$x_0 = 1$；(4) $M = 64$，$A = 11$，$x_0 = 2$。

2. 分别按以下 3 组参数用混合同余法产生 10 个伪随机数，x_1，x_2，…，x_{10}，指出哪组参数设置是正确的，说明不能正确产生伪随机数的参数组的问题出在何处。

(1) $M = 64$，$A = 13$，$C = 7$，$x_0 = 1$；(2) $M = 64$，$A = 20$，$C = 7$，$x_0 = 1$；(3) $M = 64$，$A = 13$，$C = 8$，$x_0 = 1$。

3. 试用逆变法设计出如图 2.4 所示的密度函数 $f(x)$ 的伪随机数序列的产生方法。

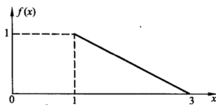


图 2.4　题 3 图

4. 试设计出几何分布的伪随机数的产生方法。其密度函数和分布函数分别为 $f(x=k)=pq^{k-1}$，$p+q=1$，$k=1,2,\cdots$和 $F(x)=1-q^{x}$。

5. 除了本书介绍的方法之外，试举出其他产生均匀伪随机数的方法，并分析不同方法的优缺点。

6. 除了本书介绍的方法外，试举出其他产生已知密度函数或分布函数的随机数的方法，并分析不同方法的优缺点。

7. 对于乘同余法，你能否想出更好的参数配置方案，从而获得更长的不重复的随机数序列长度。

参考文献

[1] Proykova A. How to improve a random number generator [J]. Computer Physics Communications, 2000, 124 (2-3): 125-131.

[2] Sanchez S, Criado R, Vega C. A generator of pseudo-random numbers sequences with a very long period [J]. Mathematical and Computer Modeling, 2005, 42 (7-8): 809-816.

[3] 李世刚，刘辉，陈标华. 超素数法长周期伪随机数发生器的应用算法 [J]. 北京化工大学学报，2003，30 (6)：1-5.

[4] 穆德　A　M，格雷比尔　F　A. 统计学导论 [M]. 史定华译. 北京：科学出版社，1978.

[5] 孙淑琴，林君，张秉仁，罗军. 伪随机序列发生器的研究与实现 [J]. 吉林大学学报（信息科学版），2004，22 (3)：185-188.

[6] 王红卫. 建模与仿真 [M]. 北京：科学出版社，2002.

[7] 赵玮，王荫清. 随机运筹学 [M]. 北京：高等教育出版社，1993.

第3章 遗传算法

遗传算法是智能优化方法中应用最为广泛也最为成功的算法。本章首先从介绍遗传算法的产生发展与基本原理开始，然后讨论遗传算法的基本构成和模板理论，介绍由不同编码方式、选择压力的调整、不同标定方法等带来的各种变形算法。最后介绍几个典型应用的例子。

3.1 导言

遗传算法（Genetic Algorithms，简称 GA 或 GAs）是由美国密歇根大学的 John H. Holland 教授及其学生于 20 世纪 60 年代末到 70 年代初提出的。在 1975 年出版的《自然与人工系统的自适应性》（Adaptation in Natural and Artificial Systems）一书中，Holland 系统地阐述了遗传算法的基本理论和方法，提出了对遗传算法的理论发展极为重要的模板理论（Schema Theory）。

后来 De Jong 和 Goldberg 等人做了大量的工作，使遗传算法更加完善。近年来，由于遗传算法求解复杂优化问题的巨大潜力及其在工业工程、人工智能、生物工程、自动控制等各个领域的成功应用，该算法得到了广泛的关注。可以说，遗传算法是目前为止应用最为广泛和最为成功的智能优化方法。

生物在自然界中的生存繁衍，显示了其对自然环境的优异的自适应能力。遗传算法所借鉴的生物学基础就是生物的进化和遗传。

3.1.1 生物的进化

生物在其延续生存的过程中，逐渐适应其生存环境，使得其品质不断得到改良，这种生命现象称为进化（Evolution）。生物的进化是以集团的形式共同进行的，这样的一个团体称为群体（Population），组成群体的单个生物称为个体（Individual），每个个体对其生存环境都有不同的适应能力，这种适应能力称为个体的适应度（Fitness）。按照达尔文的进化论，那些具有较强适应环境变化能力的生物个体具有更高的生存能力，容易存活下来，并有较多的机会产生后代；相反，具有较低生存能力的个体则被淘汰，或者产生后代的机会越来越少，直至消亡。达尔

文把这一过程和现象叫做“自然选择，适者生存”。通过这种自然的选择，物种将逐渐地向适应于生存环境的方向进化，从而产生优良的物种。

3.1.2 生物的遗传和变异

生物从其亲代继承特性或性状，这种生命现象就称为遗传（Heredity），研究这种生命现象就称为遗传学（Genetics）。由于遗传的作用，使得人们可以种瓜得瓜、种豆得豆，也使得鸟仍然在天空中飞翔，鱼仍然在水中遨游。构成生物的基本结构和功能单位是细胞（Cell）。细胞中含有一种微小的丝状化合物，称为染色体（Chromosome），生物的所有遗传信息都包含在这个复杂而又微小的染色体中。染色体主要由蛋白质和脱氧核糖核酸（DNA）组成。控制生物遗传的物质单元称为基因（Gene），它是有遗传效应的 DNA 片段。生物的各种性状由其相应的基因所控制。

细胞在分裂时，遗传物质 DNA 通过复制（Reproduction）而转移到新产生的细胞中，新细胞就继承了旧细胞的基因。有性生物在繁殖下一代时，两个同源染色体之间通过交叉（Crossover）而重组，即两个染色体的某一相同位置处的 DNA 被切断，其前后两串分别交叉组合而形成两个相同的染色体。另外，在进行复制时，可能以很小的概率产生某些差错，从而使 DNA 发生某种变异（Mutation），产生出新的染色体。

生物进化的本质体现在染色体的改变和改进上，生物体自身形态和对环境适应能力的变化是染色体结构变化的表现形式。自然界的生物进化是一个不断循环的过程。在这一过程中，生物群体也就不断地完善和发展。可见，生物进化过程本质上是一种优化过程，在计算科学上具有直接的借鉴意义。

3.2 遗传算法的基本原理

3.2.1 基本思想

遗传算法是根据问题的目标函数构造一个适值函数（Fitness Function），对一个由多个解（每个解对应一个染色体）构成的种群进行评估、遗传运算、选择，经多代繁殖，获得适应值最好的个体作为问题的最优解。具体可以描述如下。

1. 产生一个初始种群

遗传算法是一种基于群体寻优的方法，算法运行时是以一个种群在

搜索空间进行搜索。一般是采用随机方法产生一个初始种群。也可以使用其他方法构造一个初始种群。

2. 根据问题的目标函数构造适值函数（Fitness Function）

在遗传算法中使用适值函数来表征种群中每个个体对其生存环境的适应能力，每个个体具有一个适应值（Fitness Value）。适应值是群体中个体生存机会的唯一确定性指标。适值函数的形式直接决定着群体的进化行为。适值函数基本上依据优化的目标函数来确定。为了能够直接将适值函数与群体中的个体优劣相联系，在遗传算法中适应值规定为非负，并且在任何情况下总是希望越大越好。

3. 根据适应值的好坏不断选择和繁殖

在遗传算法中自然选择规律的体现就是以适应值的大小决定的概率分布来进行选择。个体的适应值越大，该个体被遗传到下一代的概率越大；反之，个体的适应值越小，该个体被遗传到下一代的概率也越小。被选择的个体两两进行繁殖。繁殖产生的个体组成新的种群。这样的选择和繁殖的过程不断重复。

4. 若干代后得到适应值最好的个体即为最优解

在若干代后，得到的适应值最好的个体所对应的解即被认为是问题的最优解。

3.2.2 构成要素

这里将遗传算法的构成要素简要说明如下，其具体的技术实现细节将在后面详细进行讨论。

1. 种群和种群大小

种群是由染色体构成的。每个个体就是一个是染色体，每个染色体对应着问题的一个解。种群中个体的数量称为种群大小或者种群规模（Population Size，Pop - Size），后文中将用 *NP* 来表示。种群规模通常是采用一个不变的常数。一般来说，遗传算法中种群规模越大越好，但是种群规模的增大也将导致运算时间的增大，一般设为 100 ~ 1 000。在一些特殊情况下，群体规模也可能采用与遗传代数相关的变量，以获取更好的优化效果。

2. 编码方法（Encoding Scheme）

编码方法也称为基因表达方法（Gene Representation）。在遗传算法中，种群中的每个个体，即染色体是由基因构成的。所以染色体与要优化的问题的解如何进行对应，就需要通过基因来进行表示，即对染色体进行正确的编码。正确地对染色体进行编码来表示问题的解是遗传算法

的基础工作，也是最重要的工作。

3. 遗传算子（Genetic Operator）

遗传算子包括交叉（Crossover）和变异（Mutation）。遗传算子模拟了每一代中创造后代的繁殖过程，是遗传算法的精髓。

交叉是最重要的遗传算子，它同时对两个染色体进行操作，组合二者的特性产生新的后代。交叉的最简单方式是在双亲的染色体上随机地选择一个断点，将断点的右段互相交换，从而形成两个新的后代。这种方法对于二进制编码最合适。遗传算法的性能在很大程度上取决于采用的交叉运算的性能。双亲的染色体是否进行交叉由交叉率来进行控制。

交叉率（记为 P_c）定义为各代中交叉产生的后代数与种群中的个体数的比。显然，较高的交叉率将达到更大的解空间，从而减小停止在非最优解上的机会；但是交叉率太高，会因过多搜索不必要的解空间而耗费大量的计算时间。

变异是在染色体上自发地产生随机的变化。一种简单的变异方式是替换一个或者多个基因。在遗传算法中，变异可以提供初始种群中不含有的基因，或者找到选择过程中丢失的基因，为种群提供新的内容。染色体是否进行变异由变异率来进行控制。

变异率（记为 P_m）定义为种群中变异基因数在总基因数中的百分比。变异率控制着新基因导入种群的比例。若变异率太低，一些有用的基因就难以进入选择；若太高，即随机的变化太多，那么后代就可能失去从双亲继承下来的好特性，这样算法就会失去从过去的搜索中学习的能力。

4. 选择策略

选择策略是从当前种群中选择适应值高的个体以生成交配池的过程。使用最多的是正比选择策略。选择过程体现了生物进化过程中“适者生存，优胜劣汰”的思想，并保证优良基因遗传给下一代个体。

5. 停止准则（Stopping Rule/Criterion）

一般使用最大迭代次数作为停止准则。

3.2.3 算法流程

下面将以 Holland 的基本 GA 为例说明算法的具体实现。Holland 的基本 GA 流程图如图 3.1 所示。

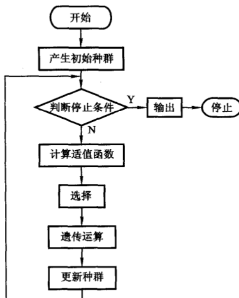


图 3.1 遗传算法流程图

由图 3.1 可以看到遗传算法实现中的各个步骤包括：(1) 初始种群的产生；(2) 编码方法；(3) 适值函数；(4) 遗传运算；(5) 选择策略；(6) 停止准则。下面就对各个步骤的实现进行具体说明。

1. 初始种群的产生

初始种群是随机产生的，具体的产生方式依赖于编码方法。种群的大小依赖于计算机的计算能力和计算复杂度。例如，**0－1** 编码的具体产生方式如下：

随机产生 $\xi_i \in U(0, 1)$，若 $\xi_i > 0.5$，则 $x_i = \mathbf{1}$

若 $\xi_i \leqslant 0.5$，则 $x_i = \mathbf{0}$

2. 编码方法——二进制编码

每个染色体可以表示为

$$\boldsymbol{X} = (x_1, x_2, \cdots, x_n), 1 \leqslant i \leqslant n$$

染色体的每一位，即 x_i是一个基因。每一位的取值称为位值。n 称为染色体的长度。Holland 的基本 GA 使用二进制编码，即使用固定长度的 **0**，**1** 字符串表示一个染色体，例如

$$\boldsymbol{X} = (\mathbf{0110010})$$

就可以表示一个染色体，该个体的染色体长度为 $n = 7$。

二进制编码适用于以下三种情况。

(1) 背包问题

n 个物品，对物品 i，价值为 p_i，质量为 w_i，背包容量为 W。如何选取物品装入背包，使背包中的物品的总价值最大。

其编码如下所示：

$$x_i = \begin{cases} 1, & \text{装入物品 } i \\ 0, & \text{不装入物品 } i \end{cases}$$

(2) 实优化问题

当精度要求高时编码长，一般不采用此编码方法，而采用实数编码。

(3) 指派问题

指派问题是一类特殊的线性规划问题，其中工作对资源的需求是一对一的。每样资源（如雇员、机器、时间段）唯一地指派给一件工作（如任务、位置、事件）。资源 $i(i=1, 2, \cdots, n)$ 指派给工作 $j(j=1, 2, \cdots, n)$ 产生一个相应费用 c_{ij}，问题的目标是如何指派可使总费用达到最小。

用 **0**，**1** 编码的变量 x_{ij}来表达资源与工作的关系如下：

$$x_{ij} = \begin{cases} 1, & \text{资源 } i \text{ 指派给工作 } j \\ 0, & \text{其他} \end{cases}$$

二进制编码缺点：编码长不利于计算。

二进制编码优点：便于位值计算，包括的实数范围大。

3. 适值函数

前面已经说明，适值函数一般根据目标函数来进行设计。目标函数一般表示为 $f(x)$，适值函数一般表示为 $F(x)$。从目标函数 $f(x)$ 映射到适值函数 $F(x)$ 的过程称为标定（Scaling）。采用方法介绍如下。

对于求目标函数最小值的优化问题，理论上只需简单加一个负号就可以将其转化为求目标函数最大值的优化问题，即

$$F(x) = -\min f(x)$$

对于求目标函数最大值的优化问题，当目标函数总为正值时，可以直接设定其适值函数就等于目标函数，即

$$F(x) = \max f(x)$$

4. 遗传运算

遗传运算，即交叉和变异，是遗传算法的精髓，也是变化最多的地方。下面就来介绍交叉和变异的具体实现。

(1) 交叉

交叉操作中，使用最多的是单点交叉和双点交叉。

①单切点交叉

单切点交叉是由 Holland 提出的最基础的一种交叉方式。从种群中选出两个个体 P_1 和 P_2，随机选择一个切点（Cutting），将切点两侧分别看作两个子串，将右侧的子串分别交换，则得到两个新的个体 C_1 和 C_2，如图 3.2 所示。

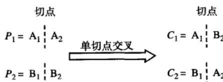


图 3.2 单切点交叉示意图 1

这里，P_1 和 P_2 称为父代染色体，C_1 和 C_2 称为子代染色体。图 3.3 是一个具体的数值例子。

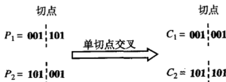


图 3.3 单切点交叉示意图 2

显然，切点的位置范围应该在第一个基因位之后，最后一个基因位之前。即设染色体长度为 n，则切点的取值范围为 $[1, n-1]$。

单切点交叉操作的信息量比较小，交叉点位置的选择可能带来较大偏差。并且染色体的末尾基因总是被交换。在实际应用中采用较多的是双切点交叉。

②双切点交叉

对于两个选定的染色体 P_1 和 P_2，随机选取两个切点，交换两个切点之间的子串，如图 3.4 所示。



图 3.4 双切点交叉示意图 1

图 3.5 是一个具体的数值例子。

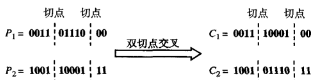


图 3.5 双切点交叉示意图 2

在算法的构成要素中已经说明，并不是所有的被选中的父代都要进行交叉操作，要设定一个交叉概率 P_c，一般取为一个较大的数，比如 0.9。

（2）变异

变异是在种群中按照变异概率 P_m 任选若干基因位改变其位值，对于 **0－1** 编码来说，就是反转位值。变异实际上是子代基因按照小概率扰动产生的变化。所以，变异概率一般设定为一个比较小的数，在 5% 以下。

5. 选择策略

最常用的选择策略是正比选择（Proportional Selection）策略，即每个个体被选中进行遗传运算的概率为该个体的适应值和群体中所有个体适应值总和的比例。对于个体 i，设其适应值为 F_i，种群规模为 NP，则该个体的选择概率可以表示为

$$P_i = \frac{F_i}{\sum_{i=1}^{NP} F_i} \tag{3.1}$$

得到选择概率后，采用旋轮法（Roulette Wheel）来实现选择操作。令

$$PP_0 = 0 \tag{3.2}$$

$$PP_i = \sum_{j=1}^{i} PP_j \tag{3.3}$$

共转轮 NP 次。每次转轮时，随机产生 $\xi_k \in U(0, 1)$，当 $PP_{i-1} \leqslant \xi_k < PP_i$，则选择个体 i。

图 3.6 是当 $NP = 10$ 时的示意图。整个转轮被分为大小不同的扇面，分别对应着不同的个体。各个个体的适应值在整个种群的全部个体的适应值之和中所占比例不同，这些比例值占据了整个转轮。较高适应值的个体对应着较大圆心角的扇面，而较小适应值的个体对应着较小圆心角的扇面。显然，旋转着的转轮停在的那一个扇面上的概率与其圆心角成正比。

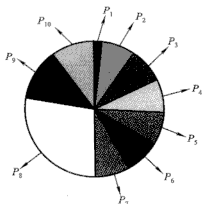


图3.6 旋轮法示意图

从图3.6中可以看到，P_8的适应值较高，作为优良的个体，其将获得较多的繁殖机会，后代很像P_8。而P_1很可能失去繁殖的机会。

6. 停止准则

遗传算法的停止准则一般是采用设定最大代数的方法，最大代数常表示为NG（Number of Max Generation）。

3.2.4 解空间与编码空间的转换

通过前面的说明已经知道，在遗传算法中需要对染色体进行编码，使一个染色体对应优化问题的一个解。例如一个问题的解为整数50，可以表示为下面的二进制编码

$$X = (\mathbf{0110010})$$

编码和解码的过程可以表示为如图3.7所示的过程。



图3.7 编码和解码

在遗传算法运行时，遗传运算是对编码后的染色体进行操作，即在编码空间内操作的。而对染色体进行评估与选择要在解空间进行。交替地在编码空间和解空间进行操作是遗传算法的一个显著特点。所以要进行两个空间的转换，如图3.8所示。

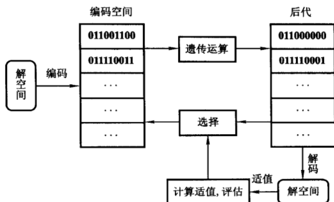


图 3.8 编码空间与解空间的转换

从编码空间到解空间的映射可能是三种情况：

(1) 1－1 映射

(2) $n-1$ 映射

(3) $1-n$ 映射

显然，1－1 映射是最好的编码方式。

3.2.5 计算举例

下面以一个简单的例子来说明遗传算法是如何工作的。

1. 最优化问题

求解以下的无约束优化问题

$$\max f(x) = x^3 - 60x^2 + 900x + 100 \tag{3.4}$$
$$x \in [0,\ 30]$$

目标函数的二维图形如图 3.9 所示。

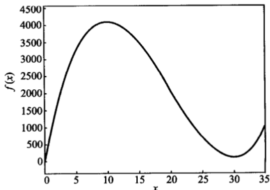


图 3.9 目标函数

2. 简单分析

首先要将决策变量编码为二进制串形式的染色体。染色体的长度取决于编码精度。设染色体长度为 L，问题的定义域为 $[a, b]$，则编码精度 C 可以表示为

$$C = \frac{b - a}{2^L - 1} \tag{3.5}$$

对于本例题，如果要求编码精度为1，则可计算染色体长度如下：

$$\frac{30 - 0}{2^L - 1} \leqslant 1$$

从而得到 $L \geqslant 5$。

所以取染色体长度为5，即可满足精度要求。

下面，用求导的方法来分析该问题的解。令

$$f'(x) = 3x^2 - 120x + 900 = 3(x - 10)(x - 30) = 0$$

则可以得到两个极值点：$x_1 = 10$，$x_2 = 30$。

计算 $f(x)$ 的二阶导数

$$f''(x) = 6x - 120$$

当 $x_1 = 10$ 时，$f''(x) < 0$，所以为极大值点。

当 $x_2 = 30$ 时，$f''(x) > 0$，所以为极小值点。

3. 步骤

（1）产生初始种群。设定参数：种群规模 $NP = 5$，最大代数 $NG = 10$，初始时刻 $t = 0$。

（2）判断停止准则。即判断最大代数是否达到 NG。

（3）计算适应值。

（4）用旋轮法正比选择。

4. 计算生成的列表

在表3.1中列出了随机产生的初始种群及相关计算结果。表中相关符号意义如下：

j：染色体编号。

x_j：各个染色体所对应的问题的解。

$f(x_j)$：染色体 j 的目标函数值，也是其适应值。

S：种群中所有个体的适应值和。

$\bar{f}$：种群中所有个体的平均适应值。

P_j：染色体 j 被选中的概率。

表 3.1 初始种群及相关计算结果

j	编码	x_j	$f(x_j)$	P_j	PP_j
1	**10011**	19	2 399	0.206	0.206
2	**00101**	5	3 225	0.277	0.483
3	**11010**	26	516	0.044	0.527
4	**10101**	21	1 801	0.155	0.682
5	**01110**	14	3 684	0.317	0.999

$$S=\sum f(x_j)=11\ 625,\ \bar{f}=\frac{S}{5}=2\ 325,\ P_j=\frac{f(x_j)}{S}$$

用旋轮法进行正比选择，根据初始种群的适应值计算得到的概率，选择时所使用的转轮如图 3.10 所示。

对初始种群中挑选出的染色体进行遗传运算时，交叉运算采用单切点交叉，变异操作采用基本位变异操作。相关参数设置为

交叉概率：　　$P_c=0.9$

变异概率：　　$P_m=0.02$

遗传运算后得到的子种群及相关计算结果列于表 3.2 中。

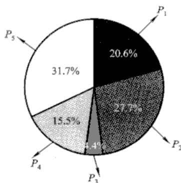


图 3.10 初始种群进行选择操作的转轮示意图

表 3.2 第一代子种群及相关计算结果

P_1	P_2	切点	变异否	编码	x_j	$f(x_j)$
1	2	4	N	**10011**	19	2 399
5	3	2	N	**01010**	10	**4 100**
5	2	3	N	**01101**	13	3 857
4	2	1	N	**10101**	21	1 801
2	5	4	N	**00100**	4	2 804

$$S=14\ 871,\ \bar{f}=2\ 992$$

5. 观察结果

从上述计算结果，可以看到

（1）整个种群在改善。种群中所有个体的平均适应值从 2 325 增长到 2 992。

（2）以 **0** 开始的编码较好，而以 **1** 开始的编码较差。

（3）“好坏”编码数量的变化。如果将以 **0** 开始编码的个体称为好的个体，用 S_1表示，而以 **1** 开始编码的个体称为差的个体，用 S_2表示，那么好坏编码的数量变化是

S_1：从 2 个增长到 3 个

S_2：从 3 个减少为 2 个

这一好坏个体的数量变化可以由表 3.1 中的数据推导得到。

$$\frac{f(S_1)}{\bar{f}}=\frac{\dfrac{f(x_1)+f(x_3)+f(x_4)}{3}}{\bar{f}}=\frac{2\ 399+516+1\ 801}{3\times 2\ 325}\approx 0.676\ 1 \tag{3.6}$$

$$0.676\ 1\times 3\approx 2$$

同理

$$\frac{f(S_2)}{\bar{f}}=\frac{\dfrac{f(x_2)+f(x_5)}{2}}{\bar{f}}=\frac{3\ 225+3\ 684}{2\times 2\ 325}\approx 1.485\ 8 \tag{3.7}$$

$$1.485\ 8\times 2\approx 3$$

可见，好坏个体数量的变化是由其适应值在整个种群中所占比例决定的。这一计算结果符合适者生存的规律。

此外，在交叉操作中，双亲的选择方法具体可以有以下几种方式。

（1）随机选择

随机产生 $\xi_k\in U(0,\ 1)$，则 $(\mathrm{Int}(\xi_k\times NP)+1)\in[1,\ 5]$，这里，Int 表示取整运算。即，可以用 $(\mathrm{Int}(\xi_k\times NP)+1)$来选择个体。

（2）比例选择

随机产生 $\xi_k\in U(0,\ 1)$，当 $PP_{i-1}\leqslant\xi_k<PP_i$ 时，选择个体 i。

（3）父亲优选而母亲随机选

3.3 模板理论

模板理论（Schema Theory）是 Holland 在早期研究中提出的，对遗传算法理论发展具有重要意义。模板理论是基于前面所讲的 Holland 的基本遗传算法。下面首先来介绍模板（Schema）的概念。

3.3.1 模板的概念

定义 3.3.1：模板是若干位取确定值，其他位不确定的一类个体的总称，用 S 来表示。

不失一般性，以二进制编码为例，个体是由二值字符集 V = {**0**，**1**} 中的元素所组成的一个字符串。而模板是由三值字符集 V_1 = {**0**，**1**，*} 中的元素所组成的字符串。其中，“*”表示通配符，它既可以被当作是 **1**，也可以被当作是 **0**。如 3.2.5 节中所提到的两类个体 S_1 和 S_2 就是两个模板。

S_1：**0** * * * *

S_2：**1** * * * *

为了对模板进行定量地描述，下面再引入模板的长度和阶数的概念。

定义 3.3.2：模板的长度 $l(S)$ 定义为模板中从左到右第一个确定位与最后一个确定位之间的长度。

定义 3.3.3：模板的阶数 $K(S)$ 定义为模板中所含有的确定基因位的个数。

例如模板 S 为：* **1** * **1** * **0** * *

则 $l(S)=4$，$K(S)=3$。

对于一些特殊的模板，如上面提到的 S_1 和 S_2 为

S_1：**0** * * * *

S_2：**1** * * * *

或者，类似下面的模板

S_3：* * **0** * * * *

由于它们只有一个确定基因位，这个位置既是第一个确定基因位，也是最后一个确定基因位，所以规定它们的长度为 1。

从上面的例子可以看出，模板的概念可以简明地描述具有相似结构的个体的编码。

下面来说明一些关于模板概念的常识。

(1) n 位编码的总长度为 $n-1$。例如，对于模板 S：**11011**，$l(S)=4$。

(2) 阶数为 $K(S)$ 的模板，模板 S 中的个体总数为 $2^{n-K(S)}$。

(3) 对于一个 n 位的二进制表达，染色体长度为 n，有

模板数 > 个体数

即

分类方法数 > 个体总数

因为模板中每一位的取值可以为**0**，**1**或者 *，所以模板总数为3^n，而个体中每一位只能取**0**或者**1**，所以个体总数为2^n。

3.3.2 模板理论

引入了模板的概念之后，遗传算法的群体进化过程可以看做是通过选择、交叉和变异运算，来不断寻找较好模板的过程。模板理论正是在分析选择、交叉和变异三个运算对模板的影响的基础之上，对模板的生存数量进行估计。模板理论由三个引理和一个主定理构成，其相关结论是在假设种群规模不变这一前提下得到的。

引理3.3.1：在正比选择下，模板S在第$t+1$代的期望个体数为

$$E(S,t+1)=f(S,t)N(S,t) \tag{3.8}$$

其中，$f(S,t)$是第t代S模板中所有个体的适应值均值与种群中所有个体的适应值均值的比。$N(S,t)$是第t代属于S模板的个体数。

例如，在3.2.5节中，对于S_1（**0** * * * *），有

$f(S_1,t)=1.4858, N(S_1,t)=2$，则$E(S_1,t+1)=1.4858\times 2\approx 3$。

证明：

种群中所有个体的适应值和为

$$Sum=\sum_{i=1}^{NP}F_i$$

则个体i的选择概率为

$$P_i=\frac{F_i}{\sum_{i=1}^{NP}F_i}$$

设NS：为S中的个体总数，且$NS=N(S,t)$，下标随着遗传的进行而变化；

$\overline{f}_S$：为S模板中的所有个体的适应值均值；

$\overline{f}$：为种群中所有个体的适应值均值。

则有

$$\begin{aligned}E(S,t+1)&=(P_1+P_2+\cdots+P_{NS})\times N\\&=\frac{F_1+F_2+\cdots+F_{NS}}{Sum}\times N\\&=\frac{\dfrac{F_1+F_2+\cdots+F_{NS}}{N(S,t)}\times N(S,t)}{\dfrac{Sum}{N}}\end{aligned}$$

$$=\frac{\overline{f}_s}{\overline{f}}\times N(S,t)$$
$$=f(S,t)\times N(S,t) \tag{3.9}$$

证毕。

可见，好的 S（均值比 $f(S,t)>1$）中的个体数在遗传运算中是不断增长的。但是，以上证明中没有考虑交叉和变异操作。而交叉和变异是可能破坏模板的，所以，要研究一个模板中的个体在交叉和变异中不被破坏的可能性。

对于使用单切点交叉操作的遗传算法有引理 3.3.2 的结论。

引理 3.3.2：第 t 代以概率 P_c 做交叉，对长度为 $l(S)$ 的模板 S 中的个体，则在第 $t+1$ 代中该个体仍为 S 中个体的概率下界为

$$P(S,t+1)\geqslant 1-\frac{P_c l(S)}{n-1}(1-P(S,t))$$

其中，$P(S,t)$ 为第 t 代个体为 S 的概率。

证明：

交叉破坏 S 的条件如下：

①进行了交叉操作，概率为 P_c。

②交叉点在模板 S 内，概率为 $\frac{l(S)}{n-1}$。

③配偶 P_2 不属于 S，概率为 $1-P(S,t)$。

所以模板 S 发生改变的最大概率为

$$\frac{P_c l(S)}{n-1}(1-P(S,t)) \tag{3.10}$$

在交叉操作中可能发生双方不属于模板 S，但是交叉后却属于模板 S 的情况，所以式（3.10）为模板 S 发生改变的最大概率。所以，模板 S 不发生改变的最小概率为

$$1-\frac{P_c l(S)}{n-1}(1-P(S,t)) \tag{3.11}$$

证毕。

对于使用简单基因位变异方式的遗传算法有引理 3.3.3 的结论。

引理 3.3.3：若第 t 代以概率 P_m 做变异，对于一个阶数为 $K(S)$ 的模板 S 中的个体，则在第 $t+1$ 代仍为 S 的概率的下界为

$$P(S,t+1)\geqslant 1-P_m\cdot K(S) \tag{3.12}$$

证明：

对于模板 S，当 $K(S)=1$ 时，不被破坏的概率为 $1-P_m$。

当$K(S)>1$时，不被破坏的概率为

$$(1-P_m)^{k(S)}$$

取其泰勒展开式的第一项，为

$$1-P_m\cdot K(S)$$

所以有

$$P(S,t+1)=(1-P_m)^{k(S)}\geqslant 1-P_m\cdot K(S)$$

证毕。

通过上面3个引理，得到了选择、交叉和变异操作对模板的影响。由此，可以综合得到下面的主定理。

主定理（模板定理）：第t代以概率P_c和P_m做交叉和变异时，长度为$l(S)$，阶数为$K(S)$，适应值均值比为$f(S,t)$的模板S在第$t+1$代的期望个体数的下界为

$$E(S,t+1)\geqslant\left[1-\frac{P_c l(S)}{n-1}(1-P(S,t))-K(S)P_m\right]f(S,t)N(S,t) \tag{3.13}$$

如果

$$f(S,t)\geqslant\left[1-\frac{P_c l(S)}{n-1}(1-P(S,t))-K(S)P_m\right]^{-1} \tag{3.14}$$

则$E(S,t)$随代数t的增加而增加。

证明：

遗传算法每一代种群要经过选择、交叉和变异操作，综合这三个操作的共同作用，由引理3.3.1，引理3.3.2和引理3.3.3可得，在第$t+1$代属于模板S的染色体数量为

$$E(S,t+1)\geqslant\left[1-\frac{P_c l(S)}{n-1}(1-P(S,t))\right](1-P_m\cdot K(S))f(S,t)N(S,t)$$

$$\geqslant\left[1-\frac{P_c l(S)}{n-1}(1-P(S,t))-K(S)P_m\right]f(S,t)N(S,t)$$

证毕。

3.4 改进与变形

前面介绍了遗传算法的基本原理，并以Holland的基本GA为例说明了算法的具体实现。在实际应用中，针对不同的问题，遗传算法可以有不同的改进和变形。这也是遗传算法内容最丰富的部分。在本节中，

将首先介绍遗传算法的实现步骤中常用的改进与变形；然后针对两类常见的优化问题，即约束优化问题和多目标优化问题，概述遗传算法对它们的解决方案。

3.4.1 编码方法

这里来介绍除了 **0-1** 编码之外的其他三种重要的编码方法。

1. 顺序编码

顺序编码是用 1 到 n 的自然数来编码，此种编码不允许重复，即 $x_i \in 1, 2, \cdots, n$，且当 $i \neq j$ 时，$x_i \neq x_j$。又称为自然数编码。例如下面是一个染色体长度为 $n=7$ 的顺序编码：

$$\boldsymbol{X} = (2\ 3\ 1\ 5\ 4\ 7\ 6)$$

对于有 7 个城市的旅行商问题，城市序号为 {1，2，…，7}，则上述编码可以表示一个行走的路线。

该编码方法具有广泛的适用范围，如指派问题、旅行商问题和单机调度等问题。但是，在使用过程中，要注意合法性问题，即是否符合所采用的编码规则的问题。满足规定的编码规则的个体称为合法的个体。如下面的编码即为一个不合法的编码：

$$\boldsymbol{X} = (2\ 2\ 1\ 5\ 4\ 7\ 6)$$

因为上述编码的前两个基因的位值相等，违反了顺序编码的规则。

2. 实数编码

对于染色体 $\boldsymbol{X} = (x_1, x_2, \cdots, x_i, \cdots, x_n)$，$1 \leq i \leq n$，$x_i \in \boldsymbol{R}$，$\boldsymbol{R}$ 为实数集，则称该染色体为实数编码。实数编码具有精度高、便于大空间搜索、运算简单的特点，特别适合于实优化问题，但是反映不出基因的特征。

3. 整数编码

对于染色体 $\boldsymbol{X} = (x_1, x_2, \cdots, x_i, \cdots, x_n)$，$1 \leq x_i \leq n_i$，$n_i$为第 i 位基因的最大取值，则称该染色体为整数编码。显然，整数编码的不同位上的基因取值可以相同。例如

$$\boldsymbol{X} = (3\ 2\ 1\ 2\ 3\ 4\ 5)$$

该染色体对于顺序编码来说是不合法的，而对于整数编码来说，是合法的。

整数编码可以适用于新产品投入、时间优化和伙伴挑选等问题。例如对于 6 个项目的伙伴挑选问题，各个项目的备选伙伴数量为

$n_1=5$，$n_2=6$，$n_3=7$，$n_4=8$，$n_5=2$，$n_6=7$

则以下即为一个合法的编码：

$$X=(2\ 2\ 4\ 5\ 1\ 7)$$

3.4.2 遗传运算中的问题

在遗传运算的过程中会遇到不合法的编码。比如图3.11所示的顺序编码，经交叉后，后代的编码不合法。

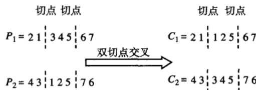


图3.11 顺序编码交叉产生不合法编码

对于这种情况，有两种应对策略：拒绝或者修复。如果使用拒绝策略，那么需要在遗传操作中出现不合法编码的比例很小。如果使用修复策略，那么可能使后代部分丢失父代的基因。

下面就来介绍相关的修复策略，来使上面的编码合法化。

1. 顺序编码的合法性修复

首先来介绍如何对顺序编码的不合法情况进行修复。根据导致不合法编码的原因，可以分为交叉修复策略和变异修复策略。

(1) 交叉修复策略

这里介绍三种主要的交叉修复策略：部分映射交叉、顺序交叉和循环交叉。

①部分映射交叉（PMX）

部分映射交叉（Partially Mapped Crossover，PMX）是用特别的修复程序来解决简单的双切点交叉引起的非法性。所以PMX包括双切点交叉和修复程序。PMX的步骤如下：

a. 选切点X，Y；

b. 交换中间部分；

c. 确定映射关系；

d. 将未换部分按映射关系恢复合法性。

以上步骤可以用图3.12来说明。如果不进行修复，那么子代 C_1 中将出现位值1，2的重复，并且丢失位值3，4；而子代 C_2 中将出现位值3，4的重复，并且丢失位值1，2。按照确定的映射关系，重复的位值被换掉，而交换的子串保持不变。

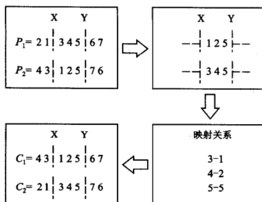


图 3.12 PMX 运算示意图

②顺序交叉（OX）

顺序交叉（Order Crossover，OX）可以看做是带有不同修复程序的PMX 的变形。OX 的步骤如下：

a. 选切点 X，Y；

b. 交换中间部分；

c. 从第二个切点 Y 后第一个基因起列出原顺序，去掉已有基因；

d. 从第二个切点 Y 后第一个位置起，将获得的无重复顺序填入。

以上步骤可以用图 3.13 来说明。与 PMX 方式得到的结果对比，可以看到，OX 相当于使用了不同的映射关系。

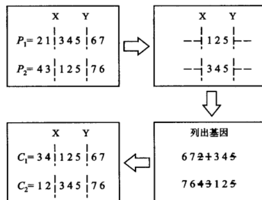


图 3.13 OX 运算示意图

OX 较好地保留了相邻关系、先后关系，满足了 TSP 问题的需要，但是不保留位值特征。

③循环交叉（CX）

循环交叉（Cycle Crossover，CX）的基本思想是子串位置上的值必须与父母相同位置上的值相等。CX的具体步骤如下

a. 选 P_1的第一个元素作为 C_1的第一位；选 P_2的第一个元素作为 C_2的第一位；

b. 在 P_1中找 P_2的第一个元素赋给 C_1的相对位置……重复此过程，直到 P_2上得到 P_1的第一个元素为止，称为一个循环；

c. 对最前的基因按 P_1、P_2基因轮替原则重复以上过程；

d. 重复以上过程，直到所有位都完成。

以上步骤可以用图3.14来说明。

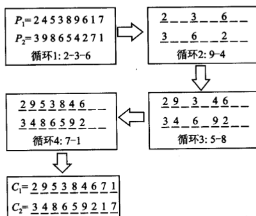


图3.14 CX运算示意图

与OX特点不同，CX较好地保留了位值特征，适合指派问题；而OX由于较好地保留了相邻关系而更适合TSP问题。

（2）变异修复策略

这里介绍两种主要的变异修复策略，换位变异和移位变异。

①换位变异

换位变异是随机地在染色体上选取两个位置，交换基因的位值，如图3.15所示。

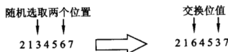


图3.15 换位变异示意图

②移位变异

移位是任意选择一位基因，将其移动到最前面，如图3.16所示。

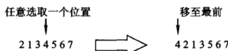


图 3.16 换位变异示意图

2. 实数编码的合法性修复

实数编码是用实向量 $X=(x_1, x_2, \cdots, x_i, \cdots, x_n)$，$1\leqslant i\leqslant n$，$x_i\in R$，来表示解的染色体。可以使用类似二进制表达的交叉和变异操作，也可以使用特殊的遗传操作，下面详细介绍。

（1）交叉

①单切点交叉

使用类似二进制表达的交叉方法对实数编码进行操作，最基本的是单切点交叉。令双亲为

$$P_1:\ X=(x_1,\ x_2,\ \cdots,\ x_n)$$

$$P_2:\ Y=(y_1,\ y_2,\ \cdots,\ y_n)$$

随机在第 k 位交叉，则可表示为如图 3.17 所示的示意图。

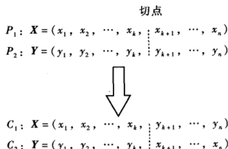


图 3.17 实数编码单切点交叉示意图

②双切点交叉

实数编码的双切点交叉与单切点交叉类似，随机选择两个切点 k，l，交换中间部分，可表示为如图 3.18 所示的示意图。

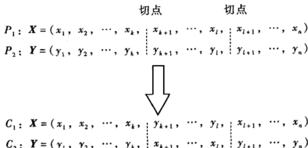


图 3.18 实数编码双切点交叉示意图

观察实数编码的单切点和双切点交叉，显然产生的子代也是合法的编码。但是，对于实数编码来说进行单切点或者双切点交叉后要注意解的可行性问题。所谓可行性是指染色体解码成为解之后是否在给定问题的可行域范围内的性质。与二进制编码类似的交叉方法在实数编码上进行操作时很容易导致解的不可行性。下面以最简单的二维实数编码为例进行说明，如图3.19所示。

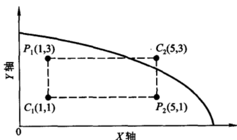


图3.19　实数编码单切点交叉导致不可行示意图

图3.19中弧形区域为问题的可行域，两个父代编码分别为：$P_1(1,3)$和$P_2(5,1)$，单切点交叉后得到两个子代为：$C_1(1,1)$ 和 $C_2(5,3)$，而C_2为不可行解。

③凸组合交叉

为了克服上面简单交叉操作导致的解的不可行性，可以使用凸组合交叉。这种运算的基本概念源于凸集理论。对于双亲

$$P_1:\ \boldsymbol{X}=(x_1,\ x_2,\ \cdots,\ x_n)$$

$$P_2:\ \boldsymbol{Y}=(y_1,\ y_2,\ \cdots,\ y_n)$$

凸组合交叉是将上述两个染色体进行如下操作：

$$\boldsymbol{Z}_1=\alpha\boldsymbol{X}+(1-\alpha)\boldsymbol{Y}$$

$$\boldsymbol{Z}_2=(1-\alpha)\boldsymbol{X}+\alpha\boldsymbol{Y}$$

即

$$\boldsymbol{Z}_1=\begin{bmatrix}\alpha x_1+(1-\alpha)y_1\\ \alpha x_2+(1-\alpha)y_2\\ \vdots\\ \alpha x_n+(1-\alpha)y_n\end{bmatrix},\boldsymbol{Z}_2=\begin{bmatrix}(1-\alpha)x_1+\alpha y_1\\ (1-\alpha)x_2+\alpha y_2\\ \vdots\\ (1-\alpha)x_n+\alpha y_n\end{bmatrix}$$

这里，$\alpha>0$。

若约束是个凸集，则可行性将得以保持。但是，这样的交叉操作将导致种群的分散性不好，基因取值向中间汇集的问题需要解决。图3.20是这个问题的一个示意图。X_1和X_4进行交叉，由于$\alpha>0$，所以得

到的子代染色体 X_1' 和 X_4' 将处于 X_1 和 X_4 之间的线段上。同样 X_2 和 X_3 进行交叉得到的子代 X_2' 和 X_3' 将处于 X_2 和 X_3 之间的线段上。新的种群继续进行同样的交叉操作，那么无论是如何进行配对，得到染色体所覆盖的区域都将越来越小。

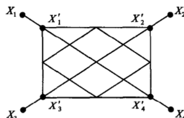


图 3.20 凸组合交叉的中间汇集问题

(2) 变异

①位值变异

二进制编码的染色体，其位值只能有 **0**，**1** 两种选择。而实数编码的染色体，其位值可以在一定范围内变化。对于实数编码的染色体 $\boldsymbol{X}=(x_1, x_2, \cdots, x_n)$，位值变异是在其基因中任选一位 $x_k(1 \leqslant k \leqslant n)$，对其进行如下变异：

$$x_k' = x_k + \delta$$

则变异后的染色体为：$\boldsymbol{Z}=(x_1, x_2, \cdots, x_k', \cdots, x_n)$

这里，δ 称为变异步长，应在一定范围内随机产生，该范围可以取为 x_k 的上下界，一般根据问题的约束域来确定。δ 可以服从均匀分布、指数分布和正态分布等。

②向梯度方向变异

对于目标函数可微的问题，实数编码的染色体也可以向梯度方向变异。

对于最大化问题，可以采用如下变异操作

$$\boldsymbol{Z}=\boldsymbol{X}+\nabla f(\boldsymbol{X}) \cdot \alpha$$
$$\alpha \in U(0, a)$$

其中，$\nabla f(\boldsymbol{X})$ 是目标函数在 $\boldsymbol{X}$ 处的梯度。

对于最小化问题，可以采用如下变异操作：

$$\boldsymbol{Z}=\boldsymbol{X}-\nabla f(\boldsymbol{X}) \cdot \alpha$$
$$\alpha \in U(0, a)$$

向梯度方向的变异使遗传算法运行时考虑到了问题本身的性质，所以效率较高。但是，染色体种群也可能因此而趋于聚集，导致种群的多

样性较差。

3.4.3 适值函数的标定

1. 标定的目的

在算法的基本原理中介绍了标定是将目标函数映射为适值函数。将个体的适应值规定为非负，从而能够直接将适值函数与群体中的个体优劣相联系。特别是对于最小化问题，必须通过标定使目标函数映射为适值函数，在任何情况下适值函数总是越大越好，理论上最简单的标定方式为

$$F(x) = -\min f(x)$$

即将其转化为最大化问题，从而实现正比选择。这可以看做是标定的第一个目的。

为了说明标定的第二个目的，首先介绍选择压力的概念。

(1) 选择压力的概念

选择压力是指种群中好、坏个体被选中的概率之差。如果差别较大，则称选择压力大。

例如：五个染色体构成的一个种群，其目标函数值分别为

$$f_1 = 1\ 010$$

$$f_2 = 1\ 008$$

$$f_3 = 1\ 002$$

$$f_4 = 1\ 005$$

$$f_5 = 1\ 015$$

则，各个染色体被选中的概率为

$$P_1 = \frac{f_1}{S} = \frac{1\ 010}{5\ 040} \approx 0.2$$

$$P_2 = \frac{f_2}{S} = \frac{1\ 008}{5\ 040} \approx 0.2$$

$$P_3 = \frac{f_3}{S} = \frac{1\ 002}{5\ 040} \approx 0.2$$

$$P_4 = \frac{f_4}{S} = \frac{1\ 005}{5\ 040} \approx 0.2$$

$$P_5 = \frac{f_5}{S} = \frac{1\ 015}{5\ 040} \approx 0.2$$

显然在选择过程中，由于目标函数之间的相对差别很小，所以各个染色体被选中的概率差别很小，即选择压力小，这将导致遗传算法的选

优功能被弱化。所以要对目标函数进行标定，来调节选择压力。这是要对目标函数进行标定的第二个目的。对于以上例子，可以进行如下标定：

$$F = f - 1\ 000$$

则各个染色体的适值分别为

$$F_1 = f_1 - 1\ 000 = 10$$

$$F_2 = f_2 - 1\ 000 = 8$$

$$F_3 = f_3 - 1\ 000 = 2$$

$$F_4 = f_4 - 1\ 000 = 5$$

$$F_5 = f_5 - 1\ 000 = 15$$

标定后各个染色体被选中的概率为

$$P'_1 = \frac{F_1}{S'} = \frac{10}{40} = 0.25$$

$$P'_2 = \frac{F_2}{S'} = \frac{8}{40} = 0.2$$

$$P'_3 = \frac{F_3}{S'} = \frac{2}{40} = 0.05$$

$$P'_4 = \frac{F_4}{S'} = \frac{5}{40} = 0.125$$

$$P'_5 = \frac{F_5}{S'} = \frac{15}{40} = 0.375$$

显然标定后，各个染色体被选中的概率差别大幅度增加，即选择压力增大了。通过上面的例子可以看到，适值函数的标定可以调节选择压力的大小。而通过调节选择压力的大小能够实现遗传算法中局部搜索和广域搜索的调节。

（2）局部搜索、广域搜索与选择压力的关系

局部搜索是针对一个较小的区域进行搜索，致力于找到更好、更精确的解。而广域搜索是进行大面积的搜索，希望找到较好的解存在的区域。显然，局部搜索和广域搜索是遗传算法中的一对矛盾，只注重局部搜索很可能陷入局优，只注重广域搜索则会导致精确开发能力不强。一个效果良好的算法应该将以上二者综合考虑，即平衡好局部搜索与广域搜索的矛盾。

一般来说，遗传算法在开始时应该注重广域搜索，通过使用较小的选择压力来实现；随着迭代的进行，逐步偏重于局部搜索，通过使用较大的选择压力来实现。

2. 适值的标定方法

下面介绍几种重要的适值标定方法。

(1) 线性标定

线性标定是采用如下的标定方法：

$$F = af + b$$

其中，f为目标函数，F为标定后的适值函数。参数a和b要根据不同的问题进行设定。

①最大化问题

对于最大化问题 $\max f(x)$，可以令$a=1$，$b=-f_{\min}+\xi$，则适值函数为

$$F = f(x) - f_{\min} + \xi$$

其中，ξ是一个较小的数，目的是使种群中最差的个体仍然有繁殖的机会，增加种群的多样性。

②最小化问题

对于最小化问题 $\min f(x)$，可以令$a=-1$，$b=f_{\max}+\xi$，则适值函数为

$$F = -f(x) + f_{\max} + \xi$$

其中，ξ也是一个较小的数，其意义与最大化问题中的设置相同。

(2) 动态线性标定

线性标定中的参数随着迭代次数的增加而变化时就得到了动态线性标定。动态线性标定是最常用的一种适值标定方法。可以表达如下：

$$F = a^k f + b^k$$

其中，上标k为迭代指标，表明参数是随着迭代次数的增加而变化的。

①最大化问题

对于最大化问题，可以令$a^k=1$，$b^k=-f_{\min}^k+\xi^k$，则适值函数为

$$F = f - f_{\min}^k + \xi^k$$

其中，$f_{\min}^k$是第k代的最小目标函数值；ξ^k是一个较小的数，目的与线性标定中的ξ相同，是使种群中最差的个体仍然有一点点繁殖的机会，从而增加种群的多样性。否则最差的个体因为适值为0，必然消失在进化的过程中。ξ^k应该随着k的增大而减小，ξ^k可以采用如下的设置方式：

$$\xi^0 = M$$

$$\xi^k = \xi^{k-1} \cdot r$$

$$r \in [0.9, 0.999]$$

通过调节M和r可以实现对ξ^k的调节。

②最小化问题

对于最小化问题 $\min f(x)$，可以令 $a=-1$，$b=f_{\max}^k+\xi^k$，则适值函数为

$$F=-f+f_{\max}^k+\xi^k$$

其中，ξ^k 也是一个较小的数，其意义与最大化问题中的设置相同。

③ξ^k 对于调节选择压力的作用

ξ^k 的引入能够调节选择压力，即好坏个体选择概率的差，使广域搜索范围宽，保持种群的多样性；而局部搜索细致，保持收敛性，如图 3.21 所示。

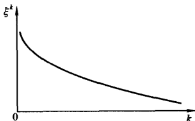


图 3.21 选择压力的调节

在算法开始运行的时候，希望选择压力较小，所以 ξ^k 取值较大，使不同个体间的选择概率相差不大；到种群进化的后期，希望选择压力较大，所以 ξ^k 取值较小，使不同个体间的选择概率相差大，种群将很快达到收敛。

(3) 幂律标定

幂律标定是采用如下的构造方式

$$F=f^{\alpha}$$

其中，α 可以用来调节选择压力：

$\alpha>1$ 时，选择压力加大；

$\alpha<1$ 时，选择压力减小。

当 α 趋近于 0 时，显然将成为随机搜索。幂律标定是比较费时的，要针对具体问题使用。

(4) 对数标定

对数标定可以用于缩小目标函数值的差别。其一般形式为

$$F=a\ln f+b$$

参数 a 和 b 根据具体问题而定。例如对于最小化问题，可以具体设置为

$$F=-\ln f+b$$

并且，$b > \ln(f_{\max})$。

（5）指数标定

指数标定的作用是扩大目标函数值的差别，其一般形式为

$$F = a\mathrm{e}^{bf} + c$$

参数 a、b 和 c 根据具体问题而定。

（6）窗口技术

采用窗口技术的适值标定方法如下：

$$F = af - f_w$$

其中，w 是窗口大小；f_w 是前 w 代中的最小目标值；w 的大小可以取为 2～10。

从表达式可以看到，实际上窗口技术也是一种动态线性标定方法。该技术的本质是将记忆引入了适值标定中，考虑到了各代 $f_{\min}$ 的波动。

（7）正规化技术

正规化技术的作用是将 f 映射到（0，1）区间，抑制超级染色体。正规化技术也是一种特殊的动态线性标定技术。对于最大化问题，标定方法如下

$$F = \frac{f - f_{\min} + r}{f_{\max} - f_{\min} + r}$$

其中，$r \in (0, 1)$。上述方法相当于对公式

$$F = a^k f + b^k$$

进行了如下设置：

$$a^k = \frac{1}{f_{\max} - f_{\min} + r}, \quad b^k = \frac{-f_{\min} + r}{f_{\max} - f_{\min} + r}$$

对于最小化问题可以标定如下：

$$F = \frac{f_{\max} - f + r}{f_{\max} - f_{\min} + r}$$

3.4.4 选择策略

传统的遗传算法的选择和遗传是一起进行的，即使后代不如父代，也无法纠正。下面介绍的选择策略都是先遗传后选择。这样，样本空间扩大了，可供选择的个体增多了。

1. 截断选择

选择最好的前 T 个个体，让每一个有 $1/T$ 的选择概率，即平均每个得到 NP/T 个繁殖机会。

例如：$NP=100$，$T=50$，那么前 50 个染色体每个的选择概率为 1/50，每个染色体平均被选中两次。

显然，这种选择策略将要花费较多时间在适应值的排序上。

2. 顺序选择

（1）顺序选择的步骤

①从好到坏排序 NP 个个体

②定义最好个体的选择概率为 q，则第 j 个个体的选择概率为

$$p(j)=q(1-q)^{j-1}$$

③由于 $\sum_{j=1}^{NP} q(1-q)^{j-1} \xrightarrow{NP\to\infty} q\frac{1}{1-(1-q)}=1$。当 NP 有限时，$\sum_{j=1}^{NP} q(1-q)^{j-1}<1$，所以要对所有个体的概率和进行归一化。令

$p_j=\bar{q}(1-q)^{j-1}$，其中 $\bar{q}=\frac{q}{1-(1-q)^{NP}}$

（2）顺序选择的特点

顺序选择的优点是选择概率可以离线计算，那么可以节省算法执行时间，并且选择压力可控；其缺点是把选择概率固定化，导致在算法的执行过程中选择压力不可调节。

（3）一个计算例子

种群中个体的概率如下

$$\text{No. }1\to p_1=q=0.1$$
$$\text{No. }2\to p_2=q(1-q)=0.09$$
$$\text{No. }3\to p_3=q(1-q)^2=0.081$$
$$\vdots$$
$$\sum \text{No.}=1$$

则可以采用旋轮法进行选择，令

$$pp_1=p_1$$
$$pp_2=p_1+p_2$$
$$pp_3=p_1+p_2+p_3$$
$$\vdots$$
$$pp_k=pp_{k-1}+p_k$$

随机产生 $\xi_k\in U(0,1)$，当 $PP_{i-1}\leqslant\xi_k<PP_i$ 时，选择个体 i。

3. 正比选择

具体实现方式同 3.2.3 节中所述，但是选择操作是在遗传操作之后进行。用动态标定来调节选择压力，采用旋轮法来共同完成种群的

选择。

3.4.5 停止准则

在基本遗传算法中，介绍了一般采用最大代数作为算法的停止准则。该方法简单易行，但是并不准确。因为可能在最大代数之前算法已经收敛，也可能在最大代数时算法还未收敛。还可以根据种群的收敛程度，即种群中适应值的一致性来判断是否算法停止。在算法的执行过程中保留历史上最好的个体，观察指标

$$\frac{\overline{F}}{F_{max}}$$

其中，$\overline{F}$ 为种群中所有个体适应值的平均值；F_{max}为所有个体适应值的最大值。当上述指标趋近于1时，说明种群收敛。同样也可以使用公式

$$|F_{max} - \overline{F}| < \varepsilon$$

$$\sum |F_i - \overline{F}| < \varepsilon$$

但是用上述判断适应值一致性的方法较为麻烦，所以较少使用。

3.4.6 高级基因操作

遗传算法中的主要遗传操作是选择、交叉和变异，它们又称为主要算子。另外还有一些人根据生物进化和遗传的机理提出了高级基因操作，或者称为次要算子，例如，倒位操作（Inversion Operation）、显性操作（Dominance Operation）、生态操作（Niche Operation）、迁移操作（Migration Operation）、性别区分（Sexual Differentiation）等。这些基因操作的研究是遗传算法理论研究中最为丰富多彩的内容。

这些基因操作来源于群体遗传学，目前应用尚少，其作用机理尚不明或没有普遍意义，还有待于进一步的研究。下面来介绍其中使用相对较多的两种操作：倒位操作和显性操作。

1. 倒位操作

倒位操作是顺序翻转染色体中两个倒位点之间的基因排列顺序，从而形成一个新的染色体。倒位操作的具体过程如下：

（1）在染色体中随机指定两个基因之后的位置为倒位点；

（2）以倒位概率 P_i顺序翻转两个倒位点之间的基因。

图3.22是对二进制编码的染色体进行倒位操作的一个示意图。

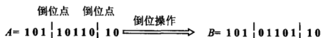


图 3.22 倒位操作示意图

如果染色体很长的话，也可以实施多段的倒位操作，如图 3.23 所示。



图 3.23 多段倒位操作示意图

倒位操作可能使染色体产生较大的变化，对于长度很大的染色体会有积极的意义，通常的交叉或者变异不易取得这种效果。

之前的编码方式往往基于如下假设：基因的功能意义与其位置是相互关联的，各个固定的位置有各自固定的功能。这种假设对于倒位操作有时是不可取的。比如生物进化过程中，各个基因的位置可能变动，但是各种基因所起的作用却是固定不变的，即基因的功能与位置是相互独立的。所以，有时需要将基因的功能与其位置互相分离，这可以通过如图 3.24 所示的染色体的扩展形式来表示。

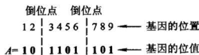


图 3.24 染色体的扩展形式

在染色体的扩展形式中，第一行是基因的位置（或者成为基因型)，第二行是基因的位值。在图 3.24 所示位置进行倒位操作之后，则得到如图 3.25 所示的新的个体。

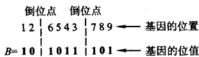


图 3.25 染色体扩展形式的倒位操作结果

从扩展形式的倒位操作结果可以看到，基因的值总是与它们原来的位置（基因型）相关，或者说各个位值保持了其原来的意义。这样，

对于扩展形式，单纯的倒位操作将不会对染色体的译码结果有影响。

倒位操作是对单个染色体进行操作，与其类似的操作还有换序操作、移序操作等，这里不再详述。

2. 显性操作

显性操作要首先从自然界中染色体的二倍体现象说起。自然界中简单生物的染色体形式是单倍体，而高等动植物的染色体形式往往是二（双）倍体或多倍体。所谓二倍体是指含有两个同源基因组的个体。如人的染色体就是23对二倍体构成的一种复杂的结构形式。二倍体结构中各个基因有显性基因和隐性基因之分。这两类基因使个体所呈现出的表现型由下述规则决定：在每个基因座上，当两个同源染色体的基因之一是显性时，则该基因所对应的性状表现为显性；而仅当两个同源染色体中对应基因全部为隐性时，该基因对应性状才表现为隐性。二倍体记忆了以前有用的基因及基因组合。显性提供了一种算子，它保护所记忆的基因免受有害选择运算的破坏。

二倍体的应用意义在于记忆能力和显性操作的鲁棒性。前者使基于二倍体结构的遗传算法能够解决动态环境下的复杂系统的优化问题，易于跟踪环境的动态变化过程；后者使得即使随机选择了适应值不高的个体，而显性操作可以利用另一同源染色体对其进行校正，从而提高运算效率，保持好的种群。

这里介绍 Holland 表述的单基因座显性映射方法。在这种方法中，描述基因的字符集为 $\{\mathbf{0}, \mathbf{1}, \mathbf{1}_0\}$，其中 $\mathbf{1}_0$ 为隐性的 **1**，**1** 为显性的 **1**。其映射关系如图 3.26 所示。

	0	$\mathbf{1}_0$	**1**
0	**0**	**0**	**1**
$\mathbf{1}_0$	**0**	**1**	**1**
1	**1**	**1**	**1**

图 3.26　单基因座显性映射方法

使用二倍体的遗传算法结构与基本遗传算法结构差别在于：

（1）显性性状也能进化，所以同源染色体之间也需进行交叉操作。

（2）变异操作考虑隐性性状。

（3）对染色体进行交叉、变异操作后要进行显性操作。

关于显性操作对于动态环境中优化问题的解决具有重要意义，在第9章中将进一步讲解。

3.4.7 约束的处理

约束优化（Constrained Optimization）是处理具有等式和（或）不等式约束的目标函数问题，是人们在实践中遇到最多的数学规划问题之一。其一般形式可以表示为

$$
\begin{aligned}
\min \quad & f(\boldsymbol{x}) \\
\text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}) \leqslant 0 \\
& \boldsymbol{h}(\boldsymbol{x}) = 0
\end{aligned}
$$

其中，$\boldsymbol{x}$ 为 $n \times 1$ 向量；$\boldsymbol{g}$ 为 m 维向量函数；$\boldsymbol{h}$ 为 l 维向量函数；f 为标量函数。

可记约束集（可行集）为

$$S = \{\boldsymbol{x} \mid \boldsymbol{g}(\boldsymbol{x}) \leqslant 0, \boldsymbol{h}(\boldsymbol{x}) = 0\}$$

一般的约束优化问题的求解难度是很大的，由于其复杂性，无论在理论研究方面还是实际应用方面都有很大难度，因此吸引了很多研究者投入其中，寻求有效的求解方法。遗传算法是其中一种常用的方法。

用于操作染色体的遗传算法常会产生不可行的染色体的问题。因此处理约束对于遗传算法解决约束优化问题非常重要。一般来说，可以将遗传算法处理约束的方法分为以下几类。

1. 拒绝策略

拒绝策略是抛弃进化过程中产生的所有不可行解。这是遗传算法处理约束问题的最简单也是效率最低的方法。当可行解不容易达到时，很难达到一个初始种群。

2. 修复策略

修复策略是在进化过程中获得不可行解后，将其修复为可行解，对于很多组合优化问题创建修复过程相对容易，但是可能导致失去种群多样性。

3. 惩罚策略

惩罚策略是对约束进行处理的最一般的方式，是通过对不可行解的惩罚来将约束问题转化为无约束问题。任何对于约束的违反都要在目标函数中添加惩罚项。这就要设计适当的惩罚函数，但是惩罚函数设计不适当则容易掩盖目标函数的优化。

4. 特殊的编码和遗传策略

也可以使用特殊的编码策略，在编码时就充分考虑约束问题，在编码时产生的都是符合约束的染色体。为了使染色体在遗传操作后仍然保持可行性，也要使用特殊的遗传策略，使遗传操作后染色体仍然保持可行。

在本书的 3.5.1 节中将以背包问题为例详细说明几种解决约束问题

的方法。

3.4.8 多目标的处理

现实的生产和生活中人们常常遇到存在的目标超过一个，并且需要同时处理的情况，而这些不同的目标又往往是相互冲突的，这就是多目标优化（Multiobjective Optimization）问题。具有 p 个目标的多目标问题的一般形式可以表示为

$$\begin{aligned}
&\min \boldsymbol{f}(\boldsymbol{x}) = \min[f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots, f_p(\boldsymbol{x})] \\
&\text{s.t. } \boldsymbol{g}(\boldsymbol{x}) \leqslant 0 \\
&\quad\ \ \boldsymbol{h}(\boldsymbol{x}) \leqslant 0
\end{aligned}$$

这里 $\boldsymbol{x}$ 为 $n\times 1$ 向量，$\boldsymbol{g}$ 为 m 维向量函数，$\boldsymbol{h}$ 为 l 维向量函数，$\boldsymbol{f}$ 为 p 维向量函数。

可记约束集（可行集）为

$$S = \{\boldsymbol{x} \mid \boldsymbol{g}(\boldsymbol{x}) \leqslant 0, \boldsymbol{h}(\boldsymbol{x}) = 0\}$$

在大多数情况下，各个目标函数间可能是冲突的。这就使得多目标优化问题不存在唯一的全局最优解，使所有目标函数同时最优。但是，可以存在这样的解：对一个或几个目标函数不可能进一步优化，而对其他目标函数不至于劣化，这样的解称之为非劣最优解。

定义 3.4.1（非劣解）：对于可行集中一个解 $\boldsymbol{x}^*$，如果找不到一个解 $\boldsymbol{x}\in S$ 使 $\boldsymbol{f}(\boldsymbol{x}) \leqslant \boldsymbol{f}(\boldsymbol{x}^*)$，则 $\boldsymbol{x}^*$ 称为非劣解。

非劣解也称有效解，或 Pareto 最优解（Pareto Optimal）。非劣解表明，在可行集中再找不到一个解比它更好。就是说，找不到一个可行解 $\boldsymbol{x}$，使得 $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_p(\boldsymbol{x}))^{\mathrm{T}}$ 的每一个目标都不比 $\boldsymbol{f}(\boldsymbol{x}^*) = (f_1(\boldsymbol{x}^*), \cdots, f_p(\boldsymbol{x}^*))^{\mathrm{T}}$ 的相应目标坏，并且 $\boldsymbol{f}(\boldsymbol{x})$ 至少有一个目标值要比 $\boldsymbol{f}(\boldsymbol{x}^*)$ 的相应目标值好。

一般非劣解不止一个，非劣解的集合称为非劣解集，用 $\boldsymbol{X}^*$ 表示。非劣解相应的目标向量称为非支配目标向量。由所有非支配目标向量构成多目标问题的非劣最优目标域。

遗传算法正越来越多地被应用于解决多目标问题，遗传算法种群进化特征使其适合于这样的问题。处理多目标问题时，遗传算法遇到的一个主要问题是如何根据多个目标函数值来确定个体的适应值，即适应值分配机制。基本的处理方法包括以下几种。

1. 向量评价方法

采用向量形式评价的适应值度量来产生下一代，而不是使用标量适应值度量方式来评价染色体。对于由 q 个目标的给定问题，每代中的选

择过程是一个循环，它重复 q 次，每次循环依次使用一个目标，每次循环使用这个目标选出下一代中的一部分个体。

2. 权重和方法

该方法为每个目标函数分配权重并将权重目标组合为单一目标函数。只需要合适的权重就可以实现该方法。权重的调整方法包括：固定权重方法；随机权重方法；适应性权重方法等。

3. 基于 Pareto 的方法

有两种基于 Pareto 的方法：Pareto 排序和 Pareto 竞争。Pareto 基于排序的适应值分配方法是希望对所有 Pareto 个体分配相同的复制概率。它主要包括两个主要步骤：

（1）基于 Pareto 排序对种群进行分类。

（2）根据排序对个体分配选择概率。

Pareto 竞争方法中采用了小生境 Pareto 概念，而不是非支配分类和排序选择。小生境 Pareto 指的是具有最小邻居数量的 Pareto 解赢得竞争。

4. 妥协方法

妥协方法是通过某种距离的度量来确定与理想解最近的解。为了克服找理想点的困难，也可以使用部分已经探索的解空间中的代理理想点来替代整个解空间的理想点。

5. 目标规划方法

该方法是采用基于排序的适应值分配方法来判断个体的价值。具体过程为：根据第一优先目标进行种群排序，如果某些个体具有相同的目标值，根据第二优先目标进行排序，如此进行下去。如果各个目标上各个个体均相同，则随机对其进行排序。然后采用从最好到最差的指数插值进行个体的适应值分配。

上述内容是对多目标处理方法的一个概述，主要来自参考文献[25]，欲了解详细内容可以查看相关参考文献。

3.5 应用实例

本节介绍遗传算法的一些应用实例。主要包括三个经典的运筹学问题：背包问题、最小生成树问题和二次指派问题，以及一个实际的应用问题：企业动态联盟中的伙伴挑选问题。解决背包问题，主要是说明遗传算法如何来处理约束；对于最小生成树问题，主要是说明遗传算法中合适的编码方法对于问题表达的重要性；而遗传算法对二次指派问题的处理可以看到

合适的编码方法能够有效地消除数学模型中的约束；最后详细介绍企业动态联盟中的伙伴挑选问题模型，给出了模型简化与编码方案，使用了嵌入模糊规则的遗传算法来求解，并给出了计算过程的详细数据。

3.5.1 背包问题

1. 问题的提出

背包问题（Knapsacks Problem）在3.2.3节中已经描述，这里为说明方便，重复如下：

n个物品，对物品i，价值为p_i，质量为w_i，背包容量为W。如何选取物品装入背包，使背包中的物品的总价值最大。

从实践的角度看，该问题可以表述成许多工业场合的应用，如资本预算、货物装载和存储分配等问题。背包问题是NP难题，适合于用遗传算法来求解。

2. 数学模型

背包问题可以用数学模型描述为

$$\max \sum_{i=1}^{n} p_i x_i \tag{3.15.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leqslant W \tag{3.15.2}$$

$$x_i = 0, 1 \quad 1 \leqslant i \leqslant n \tag{3.15.3}$$

上面的模型中采用了二进制编码方法。定义如下

$$x_i = \begin{cases} 1, & \text{装入物品 } i \\ 0, & \text{不装入物品 } i \end{cases} \tag{3.16}$$

3. 约束的处理

但是对于这样的编码方法来说，必须要面对如何保持可行性的问题。例如，对于一个7个项目的背包问题，背包容量为$W=100$，具体数据见表3.3。考察如下编码

$$X = (1\,1\,0\,0\,1\,1\,0)$$

这表示项目1、2、5和6被装入了背包，经过计算可知产生的解不可行。

表3.3 背包问题示例

i	1	2	3	4	5	6	7
w_i	40	50	30	10	10	40	30
p_i	40	60	10	10	3	20	60
p_i/w_i	1	1.2	0.33	1	0.3	0.5	2

当出现如上情况时，应该采取适当的策略来进行处理。下面将分别采用惩罚策略、解码法和顺序编码的方法来处理上面的背包问题。

(1) 罚函数法

定义适值函数为

$$F(x) = f(x)P(x) \tag{3.17}$$

其中，$f(x)$ 为目标函数；$P(x)$ 为罚函数。

令

$$P(x) = 1 - \frac{\left|\sum_{i=1}^{n} w_i x_i - W\right|}{\delta} \tag{3.18}$$

其中

$$\delta = \max\left\{W, \left|\sum_{i=1}^{n} w_i - W\right|\right\} \tag{3.19}$$

显然，当 $X = (0\ 0\ \cdots\ 0)$ 时，$\delta = W$；当 $X = (1\ 1\ \cdots\ 1)$ 时，$\delta = \left|\sum_{i=1}^{n} w_i - W\right|$。即 W 和 $\left|\sum_{i=1}^{n} w_i - W\right|$ 是 $\left|\sum_{i=1}^{n} w_i x_i - W\right|$ 的两个端点。所以上述适值函数设置的意义如下：

①δ 的作用是为了使 $0 \leqslant \left|\sum_{i=1}^{n} w_i x_i - W\right| \leqslant \delta$ 成立，保证了 $0 \leqslant P(x) \leqslant 1$。

②$P(x)$ 可行也惩罚，只有当 $\left|\sum w_i x_i - W\right| = 0$ 时不惩罚。

③罚函数的目的是将解拉向边界，尽量装满。

(2) 解码法

解码法是一段修复程序，将不合法的编码修复为合法编码。其具体步骤如下：

①将选上的物品按照 $\frac{p_i}{w_i}$ 降序排列。

②按照优先适合启发式（First Fit Heuristic）选择物品装入背包，即选前 K 个物品，使得 $\sum_{i=1}^{k} w_i x_i \leqslant W \leqslant \sum_{i=1}^{k+1} w_i x_i$。

例如对于表 3.3 中所示背包问题，如下编码

$$X = (1\,1\,0\,0\,1\,1\,0)$$

不可行。

考察该编码，被选上的物品为 1，2，5，6。

将这些物品降序排列为 2，1，6，5。

因为 $w_2 + w_1 < W < w_2 + w_1 + w_6$

所以修复为

$$X=(1\ 1\ 0\ 0\ 0\ 0\ 0)$$

(3) 顺序编码方法

使用顺序编码也可以解决背包问题，即对于 n 个问题的背包问题，使用 n 个不同的正整数代表 n 个项目。其编码步骤为：

①随机产生一个项目顺序 $(x_1, x_2, \cdots, x_n)$。

②按照优先适合启发式来选择项目，即保留项目顺序的前 n 位，使 $\sum_{i=1}^{k} w_i x_i \leqslant W \leqslant \sum_{i=1}^{k+1} w_i x_i$，从而得到可行解。

例如对于表3.3中所示背包问题，随机产生如下项目顺序（3，2，5，1，4，6，7）。

因为 $w_3+w_2+w_5<W<w_3+w_2+w_5+w_1$

所以得到的可行解为（3，2，5）。

显然顺序编码的染色体，顺序不同的时候，染色体长度也可能是不同的，那么对于编码长度可变的染色体如何进行遗传运算呢？

①交叉运算

可以使用插入交叉来进行交叉运算。其步骤如下：

a. 在第一个父代 P_1 上随机地选择一个断点。

b. 在第二个父代 P_2 上随机选择一个基因片段插入 P_1 的断点处。

c. 删去 P_1 上的重复基因。

d. 按优先适合启发式得到可行解。

图3.27为插入交叉的示意图。

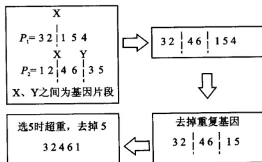


图3.27 插入交叉示意图

②变异运算

对变长的顺序编码进行变异操作可以采用如下步骤：

a. 随机删除一个基因。

b. 在染色体中随机插入一个没有的基因。

c. 对于以上原始后代用优先适合启发式方法产生一个可行解。

对于二进制编码来说，7 个项目的背包问题共有编码 $2^7 = 128$ 个，这与解空间是一一对应的，但是不能保证解的可行性；对于变长顺序编码来说，其初始编码（即随机产生的项目顺序）共有 $7! = 5\ 040$ 个，与解空间不是一一对应的，但是能够保证解的可行性。

3.5.2 最小生成树问题

最小生成树（Minimum Spanning Tree）问题是个经典的组合优化问题，这里首先描述最小生成树问题，然后介绍传统的编码方法，之后重点介绍 Prüfer 数编码的遗传算法来求解最小生成树问题。

1. 问题的提出

为描述最小生成树问题，先来说明相关的基本概念。

定义 3.5.1（图）：一个图是由点集 $V = \{v_i\}$ 和 V 中元素的无序对的一个集合 $E = \{e_k\}$ 所构成的二元组，记为 $G = (V, E)$，V 中的元素 v_i 叫做节点或端点，E 中的元素 e_k 叫做边。

定义 3.5.2（树）：连通且不含有回路的图称为树。

定义 3.5.3（生成树）：若图 G 的生成子图是一棵树，则称该树为 G 的生成树。节点的度是和该节点相连的边的数量。只有一条边相连的节点称为叶子。显然叶子的度数为 1。

树是图论中结构最简单但又十分重要的图，在自然科学和社会科学的许多领域都有广泛的应用。

定义 3.5.4（最小生成树）：连通图 $G = (V, E)$，每条边上有非负权，一棵树生成树所有边上权的和称为这个生成树的权，具有最小权的生成树称为最小生成树。

图 3.28 就是一个图和树的示意图。所有的节点和边构成了一个图。

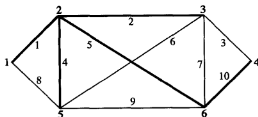


图 3.28 图和树的示意图

粗体的数字为节点，细体的数字为边。其中粗线所示的边及其所连接的节点构成了图的一棵生成树。对于这棵生成树来说，叶子节点为1、3、4和5。为了找到图的最小生成树，首先需要对树进行编码。

2. 传统的编码方法

首先来介绍传统的编码方法：节点编码方法和边编码方法。

（1）节点编码

使用树的节点的编码来表示树。例如，对于图3.28所示的树，可以表示为

{（1，2），（2，3），（2，5），（2，6），（4，6）}

（2）边编码

使用边的编码来表示树。例如，对于图3.28所示的树，可以表示为

{1，2，4，5，10}

以上两种表示方法本质上都是直观地用边来表示一棵生成树（第一种方法是用节点来表示树的边，进而来表示树）。对于一个 n 节点的图，树是连接这 n 个节点的无回路的具有 $n-1$ 条边的子图，而上面的编码方法很难避免回路，并且很难做遗传运算。

3. Prüfer 数编码

为了解决以上问题，有人提出了Prüfer数的编码方法。

（1）定义

用 $n-2$ 位自然数唯一地表达出一棵 n 个节点的生成树，其中每个数字在1和 n 之间。这样的一个排列称为Prüfer数。使用Prüfer数表示一棵树，交叉变异后还是一棵树。Prüfer数编码本质上也是节点编码的一种。

（2）应用条件

Prüfer数来表达生成树能够满足生成树的要求：

①覆盖所有节点。

②所有节点是连通的。

③没有回路。

（3）编码步骤

编码步骤如下：

①设节点 i 是标号最小的叶子。

②若边（i，j）在树上，则令 j 是编码中的第一个数字（编码顺序从左到右）。

③删去边（i，j）。

④转到①，直到剩下一条边为止。

对于图 3.28 中粗线所表示的生成树可以使用上述步骤进行编码，过程如图 3.29 所示，得到的 Prüfer 数编码为（2，2，6，2）。

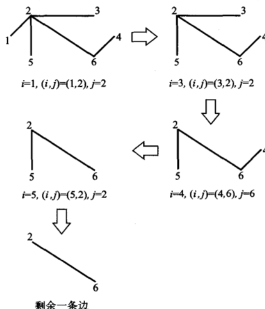


图 3.29 Prüfer 数编码示意图

（4）解码步骤

解码的步骤如下：

①令 Prüfer 数中的节点集为 P，不包含在 P 中的节点集为 $\overline{P}$。

②若 i 为 $\overline{P}$ 中最小标号的节点，j 为 P 中最左边的数字，连接边 (i, j)，并从 $\overline{P}$ 中去掉 i，从 P 中去掉 j，若 j 不再在 P 中，将 j 加入 $\overline{P}$ 中。

③重复②，直到 P 中没有节点（即 P 为空），$\overline{P}$ 中正好剩下（s，r）两个元素。

④连接（s，r）。

图 3.30 所示是上述 Prüfer 数的解码过程示意图，以刚才得到的编码（2，2，6，2）为例。

（5）优点

Prüfer 数本质上也是一种节点编码方法，它是最小生成树问题的最合适的编码方法。因为对于 n 个节点的图来说，其生成树的个数为 n^{n-2}，而 Prüfer 数的个数也为 n^{n-2}。Prüfer 数编码实现了解空间和编码

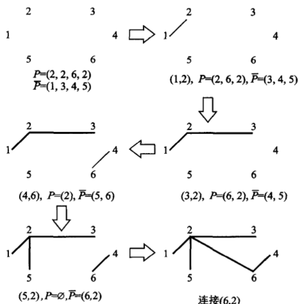


图 3.30 Prüfer 数编码示意图

空间的一一对应，并且交叉和变异运算不破坏编码的合法性。从此例可以看到，一个好的编码对于遗传算法至关重要。

Prüfer 数编码的遗传算法可以使用前述的交叉和变异方法，如简单的单点交叉和随机变异即可，这里不再详述。

3.5.3 二次指派问题

在 3.2.3 节中已经介绍了一维指派问题。这里来介绍二次指派问题（Quadratic Assignment Problem，QAP）及其遗传算法的求解。

1. 问题的提出

最早是将机器布局（Facility Layout）问题建模为二次指派问题，所以在描述二次指派问题时，常常以机器布局问题为例。该问题可描述如下：

n 台机器要布置在 n 个地方，机器 i 与 k 之间的物流量为 f_{ik}，位置 j 与 l 之间的距离为 d_{jl}，如何布置使费用最小。

二次指派问题也可以被用做许多其他不同的实际问题的模型，如大学校园中的建筑物的布局、医院中科室的安排、电子电路中的最短布线问题，以及磁带中相关数据的排序问题等。

2. 数学模型

用 0－1 编码的变量 x_{ij} 来表达机器与位置的关系如下：

$$x_{ij}=\begin{cases}1, & \text{机器 } i \text{ 布置在位置 } j \text{ 上} \\ 0, & \text{其他}\end{cases} \tag{3.20}$$

同理表示 x_{kl}。则该问题可建立二次 0 - 1 规划模型

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} x_{ij}x_{kl}f_{ik}d_{jl} \tag{3.21}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \forall j, \quad j = 1,2,\cdots,n \tag{3.22}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i, \quad i = 1,2,\cdots,n \tag{3.23}$$

$$x_{i,j}=0 \text{ 或 } 1, \ \forall i, j \tag{3.24}$$

二次指派问题是 *TSP* 问题的一般化，也是一个 *NP* 完全问题。

3. 遗传算法求解

（1）编码

可以使用顺序编码：$X=(x_1, x_2, \cdots, x_i, \cdots, x_n) \quad 1\leqslant i\leqslant n$

其中，x_i表示机器 x_i放在位置 i，x_i为 1 到 n 的整数。如编码

$X=(4, 3, 1, 2, 5)$

表示机器 4 放在位置 1；机器 3 放在位置 2；机器 1 放在位置 3；机器 2 放在位置 4；机器 5 放在位置 5。

该编码的优点是没有重复，保证了编码的合法性。

（2）目标函数表达式

使用上面的编码方式，可将目标函数简化为

$$\min \sum_{i=1}^{n}\sum_{k=1}^{n} f_{ik}d_{x_ix_k} \tag{3.25}$$

显然，目标函数变得更加简洁，更加便于计算。但是，也导致了变量出现在下标，任何数学规划不可用，而这正适合于使用遗传算法来求解。

（3）适值标定与遗传运算

可以采用下面的动态线性标定方式

$$F(x)=k[f(x)]^{-\ln[f(x)]} \tag{3.26}$$

其中，k 取 10^{12}。

选择策略使用正比选择。遗传运算可以使用循环交叉操作，变异采用换位变异。

3.5.4 企业动态联盟中的伙伴挑选问题

伙伴企业的挑选是敏捷制造和供应链管理中的一个热点问题。这里

将以该问题为例，详细介绍一个嵌入模糊规则的遗传算法的具体实现，对伙伴挑选问题该算法能够得到满意的结果。

首先将伙伴挑选问题表达为带有非解析目标函数的0-1整数规划模型，通过定义无效候选人缩小搜索空间，采用模糊规则量化的方法将模糊决策嵌入到项目调度算法中，从而形成一个模糊决策与遗传算法结合的计算方法。和传统的优化方法对比，嵌入模糊规则的遗传算法能够以很大的概率快速找到最优解。

1. 问题的提出

设某企业赢得一个由多个项目组成的大工程的标书。它本身的能力和资源不足以完成整个工程，因此它必须对项目进行招标。或者，该企业本身就是个虚拟企业，赢得标书再分发出去，从而赚取差额正是它的运作方式。

设工程由 n 个项目组成，这些项目由先后衔接关系构成了一个活动网络。若项目 k 只能在项目 i 完成后开始，即项目 i 领先于 k，则用二元关系 $(i, k) \in H$ 表示，这里，H 为所有衔接关系的集合。为描述方便，在标定项目时，使得 $i<k$，$\forall (i, k) \in H$ 成立。不失一般性，将最后一个项目标定为 n。如最后项目不能确定，可定义一个虚拟的最后项目。工程的投资流为 $e(t) \geq 0$，$t=1, 2, \cdots, D$，其中 D 为工程的交工期。若工程拖期，盟主将承受单位时间拖期惩罚 β。

对于项目 i，$i=1, 2, \cdots, n$，有 m_i 个候选人应标。项目 i 的候选人 j 的投标价格为 b_{ij}，施工周期为 q_{ij}。盟主对项目的付款方式为开工时付 αb_{ij} $(0 \leq \alpha \leq 1)$，完工时再付剩余的 $(1-\alpha) b_{ij}$。出现资金短缺时，盟主可以利率 $r>0$ 向银行借贷。

盟主的目标是选择最好的伙伴组合，使由项目开支、拖期惩罚和银行利息三者构成的总费用达到最小。

2. 数学模型

定义变量

$$w_{ij}(t) = \begin{cases} 1, \text{项目 } i \text{ 包给候选人 } j \text{ 并在时间 } t \text{ 开始} \\ 0, \text{其他} \end{cases} \tag{3.27}$$

于是，问题可用以下模型描述：

$$\begin{aligned} \min_w Z(w) = & \sum_{i=1}^{n} \sum_{j=1}^{m_i} b_{ij} \sum_{t=1}^{c_n} w_{ij}(t) + r \sum_{t=1}^{c_n} \Big[\alpha \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{\tau=1}^{t} b_{ij} w_{ij}(\tau) + \\ & (1-\alpha) \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{\tau=1}^{t} b_{ij} w_{ij}(\tau - q_{ij}) - \sum_{\tau=1}^{t} e(\tau) \Big]^{+} + \beta [c_n - D]^{+} \end{aligned} \tag{3.28}$$

$$\text{s.t.} \sum_{j=1}^{m_i}\sum_{t=1}^{c_n} w_{ij}(t) = 1, i = 1,2,\cdots,n \tag{3.29}$$

$$\sum_{j=1}^{m_i}\sum_{t=1}^{c_n}(t+q_{ij})w_{ij}(t) \leqslant \sum_{j=1}^{m_{ki}}\sum_{t=1}^{c_n} t w_{kj}(t), \forall (i,k) \in H \tag{3.30}$$

$$\sum_{j=1}^{m_n}\sum_{t=1}^{c_n}(t+q_{nj})w_{nj}(t) = c_n \tag{3.31}$$

$$w_{ij}(t) = 1 \text{或} 0, \forall i,j,t \tag{3.32}$$

其中，c_n是最后项目的完成时间；$[x]^+$表示 $\max\{0, x\}$。

由于模型［(3.28)～(3.32)］中的目标函数是非连续可微的，不能用一般的数学规划的方法来求解。

3. 存在规则解的充分条件

虽然目标函数（3.28）中没有提前惩罚项，由于借贷利息的限制作用问题一般是提前/拖期型的，但是，如果拖期惩罚因子β过小，工程会尽量拖期以避免任何借贷发生，因此，首先要考虑问题存在规则解的条件。

定义 3.5.5：若问题［(3.28)～(3.32)］的目标函数是提前/拖期惩罚型的，则称其解为规则解。

设工程的总投资为 E, $E = \sum_{t=1}^{D} e(t)$ 问题［(3.28)～(3.32)］存在规则解的充分条件由定理 3.5.1 给出。

定理 3.5.1：若$\beta \geqslant rE$，则伙伴挑选问题［(3.28)～(3.32)］存在规则解。

证明：为简化描述，设$\alpha = 1$。目标函数 $Z(w)$ 简化为

$$Z(w) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} b_{ij}\sum_{t=1}^{c_n} w_{ij}(t) + r\sum_{t=1}^{c_n}\left[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)\right]^+ + \beta[c_n - D]^+$$

设 $w(t)$, $t = 1, 2, \cdots, c_n$ 为问题［(3.28)～(3.32)］的解，而 $\overline{w}(t+1) = w(t)$, $t = 1, 2, \cdots, c_n$ 是比 $w(t)$ 延期一个单位时间的解。显然，$\overline{w}(t)$ 满足约束［(3.29)～(3.32)］且 $\overline{c}_n = c_n + 1$。两个解的目标值的差为

$$Z(w) - Z(\overline{w}) = r\sum_{t=1}^{c_n}\left[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)\right]^+ - r\sum_{t=1}^{\overline{c}_n}\left[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}\overline{w}_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)\right]^+ +$$

$$\begin{aligned}
&\beta[c_n - D]^+ - \beta[\bar{c}_n - D]^+ \\
&= r\sum_{t=1}^{c_n}[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)]^+ - \\
&\quad r\sum_{t=1}^{c_n+1}[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=0}^{t-1} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)]^+ + \\
&\quad \beta[c_n - D]^+ - \beta[c_n - D + 1]^+ \\
&= r\sum_{t=1}^{c_n}[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t} e(\tau)]^+ - \\
&\quad r\sum_{t=1}^{c_n}[\sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t+1} e(\tau)]^+ + \\
&\quad \beta[c_n - D]^+ - \beta[c_n - D + 1]^+
\end{aligned}$$

注意到 $w_{ij}(0)=0$，$\forall i, j$，可删去 $t=0$ 时的第二个 $[\cdot]^+$，因为 $e(1)\geqslant 0$，其值必为零。

定义 $y(t) = \sum_{i=1}^{n}\sum_{j=1}^{m_i}\sum_{\tau=1}^{t} b_{ij}w_{ij}(\tau) - \sum_{\tau=1}^{t+1} e(t)$，那么

$$\begin{aligned}
Z(w) - Z(\bar{w}) &= r\sum_{t=1}^{c_n}\{[y(t) + e(t+1)]^+ - [y(t)]^+\} \\
&\quad + \beta[c_n - D]^+ - \beta[c_n - D + 1]^+
\end{aligned}$$

情况1：$c_n < D$。

$\beta[c_n - D]^+ = \beta[c_n - D + 1]^+ = 0$，则有，$Z(w) - Z(\bar{w}) \geqslant 0$，因为 $[y(t) + e(t+1)]^+ \geqslant [y(t)]^+, \forall t$。

情况2：$c_n \geqslant D$。

$$\begin{aligned}
Z(w) - Z(\bar{w}) &= r\sum_{t=1}^{c_n}\{[y(t) + e(t+1)]^+ - [y(t)]^+\} - \beta \\
&\leqslant r\sum_{t=1}^{c_n} e(t+1) \leqslant rE - \beta \leqslant 0
\end{aligned}$$

定理3.5.1是符合实际的。对于投资者来说，单位时间内的工程利润至少应大于总投资的利息，而拖期惩罚显然也应大于利润。因此，定理3.5.1对于实际问题是能够满足的。

4. 模型简化与编码方案

问题［(3.28)~(3.32)］的解空间的大小，即存在解的数量 N 为

$$N = \prod_{i=1}^{n} m_i \tag{3.33}$$

显然，即使对于小规模问题这个空间也十分大。例如，对于一个

20 个项目每个项目有 5 个候选人的问题，$N=9.5367\times10^{13}$。

定义 3.5.6：项目 i 的候选人 j 是无效的，如果存在 i 的另一个候选人 k，有 $b_{ik}\leqslant b_{ij}$，$q_{ik}<q_{ij}$或者 $b_{ik}<b_{ij}$，$q_{ik}\leqslant q_{ij}$。

定理 3.5.2：若问题 [(3.28)~(3.32)] 存在规则解，则至少存在一个不含无效候选人的最优解。

证明：假设 $w'(t)$，$t=1$，2，…，c_n，是问题的最优解，其中项目 i 选择了无效候选人 j。由定义 3.5.6，必存在 i 的另一个候选人 k 满足 $b_{ik}\leqslant b_{ij}$，$q_{ik}<q_{ij}$或者 $b_{ik}<b_{ij}$，$q_{ik}\leqslant q_{ij}$。

将解 $w'(t)$ 中项目 i 的承包人 j 换为 k 得到一个新的解 $w(t)$，$t=1$，2，…，c_n。容易证明只要问题的目标函数是提前/拖期惩罚型的，必有 $Z(w)\leqslant Z(w')$。

因此，$w(t)$，$t=1$，2，…，c_n，也是最优解。

按照定理 3.5.2，求解中可以忽略所有无效候选人，而不会丢失最优解。这样，解空间可以大大简化。

对所有项目 i 将所有候选人按投标价格从低到高排序，即

$$b_{i1}<b_{i2}<\cdots<b_{im_i},i=1,2,\cdots,n \tag{3.34}$$

删去无效候选人后，有

$$q_{i1}>q_{i2}>\cdots>q_{im_i},i=1,2,\cdots,n \tag{3.35}$$

注：按定义 3.5.6，在所有候选人对中已不存在价格和时间二者之一相等的情况，而二者全等的候选人对在模型中不能区分。因此，式 (3.34) 和 (3.35) 是严格不等式。

遗传算法的染色体采用自然数编码方法，设染色体 $x=[x_1, x_2, \cdots, x_n]$，其中 x_i是 1 和 m_i，$\forall i$ 中的自然数，表示项目 i 选择候选人 x_i。$x=[x_1, x_2, \cdots, x_n]$ 也称为一个选择。例如，[3 2 5 1 4 1 6 2] 是一个 8 项目问题的染色体，或一个选择。其中，项目 1 选择候选人 3，项目 2 选择候选人 2，项目 3 选择候选人 5，以此类推。

令 $s_i(x)$ 和 $c_i(x)$ 分别为选择 x 中项目 i 的开工和完工时间，模型 [(3.28)~(3.32)] 可简化为

$$\min_x Z(x) = \sum_{i=1}^{n} b_{ix_i} + r\sum_{t=1}^{c_n(x)}\Big[\alpha\sum_{s_i(x)\leqslant t} b_{ix_i} + (1-\alpha)\sum_{c_i(x)\leqslant t} b_{ix_i} - \sum_{\tau=1}^{t} e(\tau)\Big]^+ + \beta[c_n(x)-D]^+ \tag{3.36}$$

$$\text{s.t. } c_i(x)\leqslant s_k(x),\forall(i,k)\in H, \tag{3.37}$$

$$x_i \text{ 是 1 和 } m_i \text{ 之间的整数，} i=1, 2, \cdots, n \tag{3.38}$$

模型 [(3.36)~(3.38)] 比原问题大为简化，但下标中含有变量，

这对一般数学规划方法是无法处理的，但对遗传算法来说并不十分困难。

对于一对选择 x 和 y，$x>y$ 表示 $x_i \geqslant y_i$，$\forall i$，且至少存在 1 和 n 之间的一个 k 使得 $x_k>y_k$ 成立。定义选择 x 的净值，$V(x)$，为所有项目费和借贷利息的和，即

$$V(x)=Z(x)-\beta[c_n-D]^+ \tag{3.39}$$

即拖期惩罚不包含在净值中。再定义项目 i 取候选人 h 时的费用和工期的增量 Δb_{ih} 和 Δq_{ih}，为

$$\Delta b_{ih}=b_{i(h+1)}-b_{ih},\ h=1,\ 2,\ \cdots,\ m_i-1,\ i=1,\ 2,\ \cdots,\ n$$

$$\Delta q_{ih}=q_{i(h+1)}-q_{ih},\ h=1,\ 2,\ \cdots,\ m_i-1,\ i=1,\ 2,\ \cdots,\ n$$

由不等式（3.34）和（3.35），显然有 $\Delta b_{ih}>0$ 和 $\Delta q_{ih}<0$，$\forall i,\ h$。

定理 3.5.3：对于一对选择 x 和 y，且 $x>y$，则有 $c_n(x)\leqslant c_n(y)$ 和 $V(x)\geqslant V(y)$，只要

$$\Delta b_{ih}+rb_{ih}\Delta q_{ih}\geqslant 0,\ \forall i,\ h \tag{3.40}$$

证明：设 x^2 和 x^1 是一对选择，满足 $x^2>x^1$ 且 $x_i^2-1=x_i^1=h$ 仅对 i 成立，而 $x^2=x^1$，$\forall k\neq i$。

由于总完工期 $c_n(x)$ 是活动网络的关键路径上所有项目工期的和，所以

（1）由于 $q_{i(h+1)}<q_{ih}$，容易证明必有：$c_n(x^2)\leqslant c_n(x^1)$，当项目 i 在 x^1 或 x^2 的关键路径上时不等号成立。

（2）由于 x^1 比 x^2 最大可能的提前是 $q_{ih}-q_{i(h+1)}$，而最大可能多借贷的资金为 rb_{ih}（$q_{ih}-q_{i(h+1)}$）。于是有

$$V(x^1)-V(x^2)\leqslant b_{ih}-b_{i(h+1)}+rb_{ih}(q_{ih}-q_{i(h+1)})=-\Delta b_{ih}-rb_{ih}\Delta q_{ih}\leqslant 0$$

因此，定理对 $x^2>x^1$ 成立。从 $x^1=y$ 开始，对各项目逐次加 1，记为 x^2，直到 $x^2=x$ 为止，于是定理得证。

由于 r 通常较小，条件（3.40）对绝大多数实际问题是容易满足的。定理 3.5.3 为算法中模糊规则成立的必要条件。

一旦当前选择 x 选定了所有项目的候选人，可用关键路径法（CPM）来做项目调度。CPM 除可以获得当前选择的目标值外，还能找出工程的所有关键工作和非关键工作。注意，由于候选人的工期不同，选择关键路径也不同。

5. 嵌入模糊决策的遗传算法

嵌入模糊决策的遗传算法的基本思想是在算法的第一级用遗传算法选择伙伴组合，而在第二级用模糊决策方法对选定的组合进行改进。

模糊决策的基本思想是：根据人的经验，当选择组合造成工程拖期时，对关键路径上的项目选标号较大的候选人；而工程费用较高时则可将非关键路径上的项目换为标号较小的候选人。

本节遗传算法的编码方案如前所述，适值函数按下式定义

$$f(j) = Z_{\max} - Z(j) + a, j = 1, 2, \cdots, NP$$

其中，NP 是种群大小；$Z_{\max} = \max \{Z(x(j)), j = 1, 2, \cdots, NP\}$ 为种群中的最大目标值；a 是一个动态调节选择压力的参数，它使最差的染色体也有一定的繁殖机会。a 的初值设为初始种群中的最大目标值的 5%，并在迭代中按 $a \to 0.975a$ 衰减。

遗传算子采用常用的双切点交叉和换值变异。只要变异更换的值仍在 $[1, m_i]$ 中选取，这两种遗传运算都能保持解的合法性。选择策略则采用了常用的“旋轮”正比选择，停止准则为达到指定的最大代数。算法的具体步骤见文献 [17]。

6. 计算结果与分析

下面介绍一个源于某火电站建设工程的项目发标的例子（注：数据作了处理）。

该工程由 16 个项目构成。交工期为 36 个月。拖期一个月的惩罚是 48 百万元。项目费的支付方式为开工时支付 65%，剩余部分完工时支付。银行的贷款利息是月息 0.6%。项目衔接关系如图 3.31 所示的活动网络图。

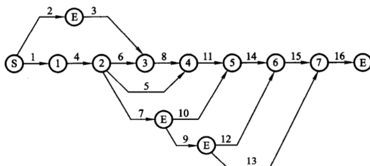


图 3.31 项目衔接关系的活动网络图

总投资为 485 百万元。由于 $0.006 \times 485 < 48$，问题有规则解。问题的解空间为 2.787×10^9，按定理 3.5.2，删去了 9 个无效候选人后，解空间降为 3.359×10^8。余下的候选人满足条件式 (3.40)。

算法获得的最好解的工程总价格为 472.484 2 百万元，完工期为 36 个月。盟主企业可获利 12.52 百万元。对比分枝定界法取得的结果，这

个解就是最优解。

为测试嵌入模糊决策的遗传算法（GA/FD）的性能，随机产生若干个不同规模的问题，并分别用（GA/FD）、单纯的遗传算法（GA）和分枝定界算法（B&B）求解，结果见表3.4。表中，“规模”表示问题的解空间的大小，“CPU时间”表示每次运行占用CPU的时间。

表3.4 不同算法的性能对比

n	规模	算法	CPU时间	最好率	最好值	平均值	最差值
16	3.359×10^{8}	GA/FD	14.55″	100%	472.48	472.48	472.48
		B&B	1.32″	100%	472.48	472.48	472.48
		GA	8.82″	39%	472.48	473.71	479.30
22	1.088×10^{12}	GA/FD	20.27″	100%	678.42	678.42	678.42
		B&B	15.71″	100%	678.42	678.42	678.42
		GA	11.99″	21%	678.42	682.03	694.10
27	9.404×10^{14}	GA/FD	22.54″	100%	917.68	917.68	917.68
		B&B	14.61″	100%	917.68	917.68	917.68
		GA	13.39″	2%	918.67	928.22	940.85
30	2.257×10^{16}	GA/FD	26.24″	100%	1 010.65	1 010.65	1 010.65
		B&B	30.87″	100%	1 010.65	1 010.65	1 010.65
		GA	15.16″	3%	1 010.65	1 020.46	1 036.55
38	3.510×10^{20}	GA/FD	31.99″	80%	1 362.47	1 362.83	1 366.14
		B&B	458.47″	100%	1 362.47	1 362.47	1 362.47
		GA	18.29″	1%	1 362.47	1 375.34	1 388.98
48	4.912×10^{25}	GA/FD	40.52″	78%	1 636.07	1 636.55	1 639.91
		B&B	42 457.73″	100%	1 636.07	1 636.07	1 636.07
		GA	22.88″	1%	1 639.55	1 659.86	1 682.51
60	9.168×10^{30}	GA/FD	49.31″	70%	1 870.85	1 871.61	1 883.47
		B&B	142 937.01″	100%	1 870.85	1 870.85	1 870.85
		GA	27.61″	1%	1 899.27	1 920.80	1 946.12

由表3.4可见，问题的规模随项目数的增加而快速增加。推荐的嵌入模糊决策的遗传算法能以较大的概率获得最优解，且计算时间随规模增加的增加量不大。分枝定界法能确保最优解，但对大规模问题计算时间太长。纯遗传算法虽也能解大型问题，但一般很难得到最优解。以上比较说明模糊决策能够有效地改进复杂优化问题的计算性能。

7. 结论

从上面的结果可以看到，嵌入模糊决策的遗传算法能以大的概率较

快地取得企业动态结盟中的伙伴挑选问题的最优解。比起纯遗传算法和分枝定界法，它在计算速度和达优率两方面的综合性能更好。

3.5.5 准时化生产计划的半无限规划模型

自从准时化（JIT）生产技术获得成功之后，以准时化为目标的提前/拖期生产调度问题成为一个十分活跃的研究领域。为了准确描述准时化的生产计划问题，汪定伟等用连续时间函数来描述产品对制造资源的需求，在此基础上建立了一个非线性的半无限规划模型，并开发出一种沿梯度方向变异的遗传算法。实验表明，当种群规模足够大时，该算法能够以很大的概率找到最优解。

1. 问题描述及半无限规划模型

某制造系统在计划期［0，T］内接到 n 项订货。订货 i 的制造周期是 L_i，交货期是 d_i。设 x_i是订货 i 计划的完成时间，其对制造资源的需求是依赖于 x_i和时间 t 的函数，记为 $R_i(t,\ x_i)$。定义 $G(t)$ 是 t 时刻的资源可用量，$t\in[0,\ T]$。按 JIT 思想，计划的生产完成时间应尽可能靠近交货期。于是，采用提前/拖期生产调度的二次型惩罚函数，该问题可用如下非线性的半无限规划来描述。

$$\min \sum_{i=1}^{n} a_i(x_i - d_i)^2 \tag{3.41}$$

$$(\text{SIP})\ \text{s.t.}\ \sum_{i=1}^{n} R_i(t,x_i) \leqslant G(t),t \in [0,T] \tag{3.42}$$

$$0\leqslant L_i\leqslant x_i\leqslant T,i=1,2,\cdots,n \tag{3.43}$$

其中，a_i是订货 i 的权重，一般正比于订货额。

由于 t 在［0，T］中连续取值，因此式（3.42）中有无限个约束。虽然将时间取离散值可将该问题转化为一个普通的非线性规划问题，但当离散点较多时约束个数较多，计算十分复杂。相反，用连续时间函数来描述资源需求和拥有量时，可按照文献［18］提出的非精确算法，利用连续函数的性质，只将资源需求函数的有限个极值点作约束处理，计算反而更为简便。

资源需求函数可以根据实际情况选取，一般可以取为钟形函数（正态分布密度函数），即

$$R_i(t,x_i) = a_i\exp\left[\frac{-(t-x_i+b_i)^2}{c_i}\right],i=1,2,\cdots,n \tag{3.44}$$

设 p_i是订货 i 的总资源需求，按式（3.44）中函数的性质，其参数可按下式确定：

$$b_i=\frac{L_i}{2},c_i=\frac{L_i^2}{8},a_i=P_i\ \sqrt{2\pi}\left(\frac{L_i}{4}\right),i=1,2,\cdots,n \tag{3.45}$$

假设函数（3.44）的两个标准差之外的部分（小于3%）可以忽略不计。

资源拥有量函数 $G(t)$ 一般可用指数增长函数来表示。由于设备检修或其他任务可能占用部分资源，这些已占用资源可用类似（3.44）的函数表示，并从总资源中扣除。于是有

$$G(t)=g_0\exp(\beta t)-\sum_{i=1}^{m}q_i\exp\left[-\frac{(t-g_i)^2}{h_i}\right] \tag{3.46}$$

其中，q_i、g_i、h_i 可根据检修的资源占用量和时间长短，类似于式（3.45）来确定。g_0为资源的初始值，β 为增长率。

分析表明，由于每一对订货都是相互竞争资源的，在资源有限的前提下，一个订货占据一段资源后，另一个订货只有提前或者拖后，因此规划（SIP）的可行域一般是不连通的。最坏情况分析表明，对于 n 个订货问题，可行域可能由 $2^{\frac{n(n-1)}{2}}$ 个分离的区域构成。这就使得传统的优化方法很难应用，于是遗传算法便成为优先考虑的选择。

2. 沿梯度方向变异的遗传算法

（1）基因表达方法

对于本文的问题，最方便的方法是取规划（SIP）的变量，订单完成时间 x_i 作为基因。令 N 为种群规模，对个体 j，其基因表达即为实向量

$$X(j)=[x_1(j),x_2(j),\cdots,x_n(j)]^{\mathrm{T}},j=1,2,\cdots,N \tag{3.47}$$

（2）适应值函数

由于（SIP）中有无限个约束，为此这里采用文献［5］提出的非精确算法。其基本思想是每步迭代中只处理有限个最不满足约束，直到所有约束都满足为止。

由于 $R_i\ [t,\ x_i(j)]$ 是单峰函数，约束（3.42）的右边最多只有 n 个局部最大点，且这些点位于区间 $[x_i-L_i,\ x_i]$（$i=1,\ 2,\ \cdots,\ n$）之内。

对于订单 i（$i=1,\ 2,\ \cdots,\ n$），定义

$$t_i^*=\arg\max\left\{\sum_{i=1}^{n}R_i[t,x_i(j)]-G(t)\mid x_i(j)-L_i<t<x_i(j)\right\} \tag{3.48}$$

$$\Phi[t_i^*,x(j)]=\sum_{i=1}^{n}R_i[t_i^*,x_i(j)]-G(t^*) \tag{3.49}$$

若 $\Phi(t_i^*, x(j)) > 0$，则该约束不能满足。引入资源约束的容差 $\varepsilon_c > 0$，则集合

$$VT(j) = \left\{ t_i^* \mid i = 1,2,\cdots,n, \Phi[t_i^*, X(j)] > \varepsilon_c \right\}, j = 1,2,\cdots,N \tag{3.50}$$

即为个体 j 的最不满足的约束集。个体 j 的扩展目标函数为

$$\Psi(j) = \sum_{i=1}^{n} a_i(x_i - d_i)^2 + M \sum_{t_i^* \in VT(j)} \Phi[t_i^*, X(j)], j = 1,2,\cdots,N \tag{3.51}$$

这里，M 是一个充分大的罚因子。注意，经过如上变换后，$\Psi(j)$ 已不再是时间 t 的函数。定义

$$F_{\max} = \max\{\Psi(j) \mid j = 1,2,\cdots,N\} \tag{3.52}$$

那么个体的适应值函数 $F(j)$ 可用式（3.53）计算

$$F(j) = \gamma F_{\max} - \Psi(j), j = 1,2,\cdots,N \tag{3.53}$$

这里，$\gamma > 1$，是一个不同适应值函数的个体选择概率的控制系数。实验中 $\gamma = 1.05$，这样最差的个体仍可以以一个很小的概率产生下一代。

（3）选择策略

使用旋轮法的正比选择。

（4）遗传算子

由于（SIP）的可行域是非连通的，交叉可能由可行解产生不可行解，于是变异成了唯一的选择。负梯度方向是目标函数的最速下降方向，沿负梯度方向的变异可以更快地达到可行域，并取得最优解。沿负梯度方向变异的遗传算子描述如下。

对于个体 j，其扩展的目标函数（3.51）的梯度向量为

$$\nabla \Psi(j) = [\nabla \Psi_1(j), \nabla \Psi_2(j), \cdots, \nabla \Psi_n(j)]^{\mathrm{T}} \quad j = 1,2,\cdots,N \tag{3.54}$$

其第 i 个分量为

$$\nabla \Psi_i(j) = 2a_i(x_i - d_i) + M \sum_{t_i^* \in VT(j)} \frac{2a_i(t_i^* - x_i(j) + b_i) R_i(t_i^*, x_i(j))}{c_i} \tag{3.55}$$

若按选择策略，第 $k+1$ 代的某个体（即第 k 代的孩子）选择个体 j 为其父亲，则

$$X^{k+1}(s) = X^k(j) - \rho \nabla \Psi(j), s = 1,2,\cdots,N \tag{3.56}$$

这里，ρ 是按 Erlang 分布产生的一个随机步长，上标 k 是代数指标。

由于 M 远大于 a_i，当 $VT(j)$ 非空时，$-\nabla\Psi(j)$ 主要是寻找可行域

方向，一旦达到可行域，$VT(j)$ 则变为空集，该方向变为寻找局部最优点的方向。

分析表明，在算法的初始阶段，随机产生的个体大都是不可行的，其适应值函数的大小大致相当，因此各个个体都有机会产生下一代，从而达到就近的可行域。随着算法的进行，某些个体的目标函数明显优于其他个体，它们得到更多的机会产生下一代，产生的下一代随机地分布在它们的负梯度方向上。选择策略近似为一个线性搜索程序。因此，负梯度方向变异的遗传算子，使该算法在初始阶段是一个随机抽样算法，随着迭代的进行，算法逐步变为一个多点的最速下降法。算法的这种性质是原问题所需要的。

（5）停止准则

遗传算法通常以达到最大的繁殖次数作为停止准则。对于本小节所讨论的问题，除了必须达到最大的代数外，还要检查是否达到了可行解，即检查是否满足

$$VT^{NG}(j^*) = \varnothing \tag{3.57}$$

其中，NG 是最大代数；j^* 是第 NG 代中最优个体的个数；$\varnothing$表示空集。

由于在算法的初始阶段找可行域时，步长应大一些；而在算法终止前为保证计算的精度，步长应很小。因此，产生随机步长的 Erlang 分布参数，应随着步长的增加而减小。

（6）算法步骤

算法的具体计算步骤如下。

第1步：指定 Erlang 分布的参数 m，μ 和缩减率 r。输入种群规模 N 和最大代数 NG。输入约束容差 ε_c 和计算要求精度 ξ。

第2步：按下式确定计划水平

$$T = \frac{\max\{\eta(\sum_{i=1}^{n} P_i + \sum_{j=1}^{m} q_j), \max[d_i \mid i = 1,2,\cdots,n]\}}{g_0} \tag{3.58}$$

η 是一个资源的松弛因子，可取 $\eta = 1.30$。

第3步：产生初始的种群。对于 $k=1, 2, \cdots, N$，有

$$x_i^0(j) = L_i + \xi_i(T - L_i), i = 1,2,\cdots,n \tag{3.59}$$

其中，$\xi \in U(0, 1)$，迭代指标 $k=0$。令最优个体为 $j^* = 1$。

第4步：令 $k = k+1$。如 $k > NG$，则输出停止；否则，按公式 $\mu^{k+1} = r\mu^k$ 缩小 μ。

第5步：用公式（3.51）~（3.53）计算所有个体的扩展目标函数和适应值函数，并选择

$$j' = \arg\max\{F(j) \mid j = 1, 2, \cdots, N\} \tag{3.60}$$

如果 $\Psi(j') < \Psi(j^*)$，则令 $j^* = j'$。

第 6 步：计算选择概率，并按选择策略为下一代个体选择父亲；按式（3.54）～（3.56）计算产生下一代个体。

第 7 步：更新所有个体，即

$$x^k(j) = x^{k+1}(j), \quad j = 1, 2, \cdots, N \tag{3.61}$$

转第 4 步。

3. 数值结果及分析

以上算法用 Fortran 语言编程，在计算机上计算了大量例题，取得了满意的计算结果。这里介绍一个源于实际背景的问题。

某建筑公司接到 10 份建筑工程订单。该公司的主要资源约束是人力，当前的可用人力为 100 kh/w（千时/周），该公司的人力将按 0.5% 的速率增长。有两项维修工程已经排定，其人力需求分别为从第 20 周至 40 周需 200 kh，第 30 周至 70 周需 300 kh。各订单的建筑周期、总人力需求、合同总额及希望的交货期见表 3.5 所示。公司希望根据自己的建筑能力，初排一个尽可能接近客户需求的交货期的建筑计划，以便与客户商定。

表 3.5 订单数据及初排的计划完成时间

订单号 No.	建筑周期 L_i/w	总人力需求 P_i/kh	合同总额 $a_i/10^4$ \$	需求交货期 d_i/w	计划完成时间 x_i/w
1	20	400	10	25	30.74
2	20	900	18	30	20.76
3	30	800	12	35	41.88
4	40	800	15	40	40.79
5	25	1 000	28	40	52.50
6	20	1 200	20	40	86.77
7	50	2 000	30	50	84.38
8	10	300	18	15	10.00
9	20	400	9	50	75.75
10	60	1 500	30	60	87.28

由于不少客户的交货期集中在第 40 周和 50 周，在第 20 周至 30 周形成一个不可能满足的资源需求高峰。按前面介绍的算法计算，经 737

代遗传找到一个所能得到的最好解，见表3.5最后一列所示。该结果在满足资源约束的前提下充分利用了资源。

由于问题的可行域是非凸且非连通的，用一般的非线性规划方法对扩展的目标函数（3.51）做优化，不能取得最优解，即使用遗传算法也可能终止在局部最优解上。例如，对于一个规模较小的问题，从不同的随机数种子出发，经过反复计算，算法终止在6个不同的解(A，…，E)上。

计算中，逐步增大种群规模，发现当种群规模较小时，算法终止在局部最优解的可能性较大。随着种群规模增大，算法终止在最优解的概率逐步增大，见表3.6。因此，实际应用中在不超过计算机能力的前提下，尽可能选择较大的种群规模，就能取得较好的结果。

表3.6 不同种群大小的结果比较

	1	2	3	4	5	6	7	8	9	10	寻优概率
25	E	E	F	A	C	A	E	D	C	C	0.2
50	A	A	A	A	C	A	E	A	C	B	0.6
75	E	A	A	C	A	C	A	A	A	A	0.7
100	A	A	A	A	A	A	A	A	A	A	1.0

问题与思考

1. 对于编码长度为7的0-1编码，判断以下编码的合法性。

(1) [1 0 2 0 1 1 0]

(2) [1 0 1 1 0 0]

(3) [0 1 1 0 0 1 0]

(4) [0 0 0 0 0 0 0]

(5) [2 1 3 4 5 7 6]

2. 对于编码长度为7的顺序编码，判断以下编码的合法性。

(1) [7 1 2 0 4 3 5]

(2) [1 3 6 2 4 7]

(3) [2 1 3 5 4 7 6]

(4) [8 1 4 3 2 5 7]

(5) [2 1 3 2 5 7 6]

3. 对于编码长度为7的实数编码，判断以下编码的合法性。

(1) [3.5 1.9 2 7 1.8 1.7 0]

(2) [89.05 4.78 2 1 4.3 6.9]

(3) [0 1 1 0 0 1 0]

(4) [0 0 0 0 0 0 0]

(5) [2 1 3 4 5 7 6]

4. 对于背包问题：7 件财宝的价值 p_i，质量 w_i，$i=1, 2, \cdots, 7$，参见表题 3.4。

表题 3.4

i	1	2	3	4	5	6	7
p_i	30	60	25	8	10	40	60
w_i	40	40	30	5	15	35	30

如背包的容量为 120，按适合优先启发式（First Fit Heuristic）将以下编码合法化，并计算以下种群中（Pop－Size＝5）各个个体的适值和选择概率。

(1) [6 4 3 5 7 1 2]　　(2) [7 2 4 3 5 6 1]

(3) [1 3 4 2 6 5 7]　　(4) [2 7 3 1 5 4 6]

(5) [5 3 2 4 7 6 1]

5. 双亲染色体分别为

P_1: [6 1 2 8 9 5 4 7 10 3]

P_2: [10 7 4 1 3 6 2 8 5 9]

两个切点位置分别为：4 和 8。试分别使用 PMX、OX 和 CX 产生两个子代染色体。

6. 写出如下生成树的 Prüfer 数编码。

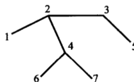


图 3.32　题图 6

7. 一个 7 节点的生成树的 Prüfer 数编码为 [6 3 2 4 4]，试画出该生成树。

参考文献

[1] Cheng R, Gen M. An adaptive superplane approach for multiple objective optimization problems [R]. Ashikaga: Ashikaga Institute of Technology, 1998.

[2] Cheng R, Gen M. Compromise approach – based genetic algorithms for bicriterion shortest path problems [R]. Ashikaga: Ashikaga Institute of Technology, 1998.

[3] Coello C A. An updated survey of evolutionary multiobjective optimization techniques: state of the art and future trends: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1999: 3 – 13.

[4] Fang S C, Wu S Y. An inexact approach to solving linear semi – infinite programming problems [J]. Optimization, 1994, 28 (2): 291 – 299.

[5] Gen M, Liu B, Ida K. Evolution program for deterministic and stochastic optimizations [J]. European Journal of Operational Research, 1996, 94 (3): 618 – 625.

[6] Goldberg D. Genetic algorithms in search, optimization and machine learning [M]. Massachusetts: Addison Wesley, 1989.

[7] Holland J H. Adaptation in natural and artificial systems [M] . Ann Arbor: University of Michigan press, 1975.

[8] Horn J, Nafpliotis N, Goldberg D. A niched pareto genetic algorithm for multiobjective optimization: Proceedings of the IEEE conference on evolutionary computation [C]. Piscataway, NJ: IEEE Press, 1994: 82 – 87.

[9] Michael Z. Genetic algorithms, numerical optimization, and constraints: Proceedings of the 6th international conference on genetic algorithms [C]. San Francisco: Mogan Kaufmannn Publishers, 1995: 151 – 158.

[10] Michalewicz Z. A survey of constraint handling techniques in evolutionary computation method: The 4th annual conference on evolutionary programming [C]. MIT Press, Cambrige, 1995: 135 – 155.

[11] Murata T, Ishibuchi H, Tanaka H. Multiobjective genetic algorithm and its application to flowshop scheduling [J]. Computers and Industrial Engineering, 1996, 30 (4): 957 – 968.

[12] Orvosh D, Davis L. Using a genetic algorithm to optimize problems with feasibility constraints: Proceedings of the IEEE Conference on evolutionary computation [C]. Piscataway NJ: IEEE Press, 1994: 548 – 552.

[13] Richardson J, Palmer M, Liepins G, Hilliard M. Some guidelines

for genetic algorithms with penalty functions: Proceedings of the 3rd international conference on genetic algorithms [C]. San francisco: Morgan Kaufmann publishers, 1989: 191 - 197.

[14] Schaffer J. Multiple objective optimization with vector evaluated genetic algorithms: Proceedings of the first international conference on Genetic algorithms [C]. Hillsdale, NJ: Lawrence Erlbaum Associates, 1985: 93 - 100.

[15] Wang D. Earliness/tardiness production planning approaches for manufacturing systems [J]. Computers & Industrial Engineering, 1995, 28 (3): 425 - 436.

[16] Wang D, Fang S. A semi - infinite programming model for earliness/tardiness production planning with a genetic algorithm. Computer & Mathematics with Applications. 1996, 31 (8): 95 - 106.

[17] Wang D W, Yung K L, Ip W H. A Heuristic Genetic Algorithm for Subcontractor Selection in a Global Manufacturing Environment [J]. IEEE Trans. On SMC Part - C, 2001, 31 (2): 189 - 198.

[18] Wang D W, Yung K L, Ip W H. Partner selection model and soft computing approach for dynamic alliance of enterprises [J]. Science in China, Series F, 2002, 45 (1): 68 - 80.

[19] Zheng D, Gen M, Cheng R. Multiobjective optimization using genetic algorithms [J]. Engineering Valuation and Cost Analysis, 1999, 2: 303 - 310.

[20] 李敏强，寇纪淞，林丹，李书全．遗传算法的基本理论与应用[M]．北京：科学出版社，2002.

[21] 汪定伟，Shu - Cherng Fang（方述诚）．准时化生产计划的半无限规划模型及其遗传算法［J］．控制与决策，1996，11（4）：446 - 451.

[22] 汪定伟，叶伟雄，容启亮．企业动态联盟中的伙伴挑选模型及其软计算方法［J］．中国科学（E辑），2002，32（6）：824 - 830.

[23] 王凌．智能优化方法及其应用［M］．北京：清华大学出版社，施普林格出版社，2001.

[24] 席裕庚，柴天佑，恽为民．遗传算法综述［J］．控制理论与应用，1996，13（6）：697 - 708.

[25] 玄光男，程润伟．遗传算法与工程设计［M］．汪定伟，唐加福，黄敏译．北京：科学出版社，2000.

[26] 玄光男，程润伟. 遗传算法与工程优化 [M]. 于歆杰，周根贵译. 北京：清华大学出版社，2004.
[27] 周明，孙树栋. 遗传算法原理及应用 [M]. 北京：国防工业出版社，1999.

第4章 禁忌搜索算法

禁忌搜索（Tabu Search 或 Taboo Search，简称 TS）算法是继遗传算法之后出现的又一种元启发式（Meta - Heuristic）优化算法，最早于1977年由 Glover 提出。禁忌搜索算法模仿人类的记忆功能，使用禁忌表来封锁刚搜索过的区域来避免迂回搜索，同时赦免禁忌区域中的一些优良状态，进而保证搜索的多样性，从而达到全局优化。迄今为止，禁忌搜索算法已经成功应用于组合优化、生产调度、机器学习、神经网络、电力系统以及通信系统等领域，近年来又出现了一些对禁忌搜索算法的改进与扩展。本章对禁忌搜索算法的基本思想、关键环节、计算流程以及基本改进与应用做一介绍。

4.1 导　言

早在1977年，Glover 就提出了禁忌搜索算法，并用来求解整数规划问题，随后又用禁忌搜索算法求解了典型的优化问题——旅行商问题（TSP）。1989—1990年 Glover 在《Operation Research of America》上系统地介绍了禁忌搜索算法以及一些成功的应用，于是禁忌搜索算法引起了广泛的关注。本节简要介绍禁忌搜索算法的产生背景和基本思想，并用一个简单算例说明禁忌搜索算法中的一些基本概念。

4.1.1 局部邻域搜索

局部邻域搜索算法基于贪婪思想，持续地在当前邻域中搜索，直至邻域中再也没有更好的解，也称为爬山启发式算法。考虑如下优化问题：

$$(\mathbf{P})\quad \min c(x): x \in X \subset R^n \tag{4.1}$$

其中，目标函数 $c(x)$ 可以是线性的或者非线性的，解空间 X 由 n 维实空间上的有限个离散点构成。实际问题中，解空间 X 可能由各种各样特定的约束条件构成。邻域搜索的过程就是从一个解移动到另外一个解，这里的移动用 s 表示，移动后得到的一个解用 $s(x)$ 表示，从当前解出发的所有移动得到的解的集合用 $S(x)$ 表示，也就是邻域的概念。简单的邻域搜索算法可以描述为

第 1 步：选择一个初始解 $x \in X$。

第 2 步：在当前解的邻域中选择一个能得到最好解的移动 s，即

$$c(s(x)) < c(x),\ s(x) \in S(x)$$

如果这样的移动 s 不存在，则 x 就是局部最好解，算法停止；否则$s(x)$是当前邻域中的最好解。

第 3 步：令 $x = s(x)$ 为当前解，转第 2 步，继续搜索。

这种邻域搜索方法容易理解，容易实现，而且具有很好的通用性，但是搜索结果完全依赖于初始解和邻域的结构，而且只能搜索到局部最优解。为了实现全局搜索，禁忌搜索采用允许接受劣解来逃离局部最优解。

4.1.2 禁忌搜索算法的基本思想

禁忌搜索算法的基本思想就是在搜索过程中将近期的历史上的搜索过程存放在禁忌表（Tabu List）中，阻止算法重复进入，这样就有效地防止了搜索过程的循环。禁忌表模仿了人类的记忆功能，禁忌搜索因此得名，所以称它是一种智能优化算法。

具体的思路如下：禁忌搜索算法采用了邻域选优的搜索方法，为了能逃离局部最优解，算法必须能够接受劣解，也就是每一次迭代得到的解不必一定优于原来的解。但是，一旦接受了劣解，迭代就可能陷入循环。为了避免循环，算法将最近接受的一些移动放在禁忌表中，在以后的迭代中加以禁止。即只有不在禁忌表中的较好解（可能比当前解差）才被接受作为下一次迭代的初始解。随着迭代的进行，禁忌表不断更新，经过一定迭代次数后，最早进入禁忌表的移动就从禁忌表中解禁退出。

下面给出一个简单的算例来说明禁忌表的作用。由七种不同的绝缘材料构成一种绝缘体，如何排列这七种材料才能使得绝缘的效果最好？绝缘效果的好坏以绝缘数值表示，绝缘数值越大，绝缘效果越好。

当算法迭代到某一步的时候，各种材料的排列顺序为 2—4—7—3—5—6—1，交换各种材料对绝缘效果的改善情况见表 4.1，其中正值表示绝缘效果变好，负值表示绝缘效果变坏。可见，交换材料 1 和 3 可以增加绝缘数值 2，对绝缘材料的改善效果最好。交换之后七种材料的排列顺序为 2—4—7—1—5—6—3，绝缘数值为 18，同时将这个交换（1，3）加入到禁忌表中。

此时，两两交换各种材料对绝缘效果的影响情况见表 4.2，可见交换任意两种材料的排列顺序都不能改善绝缘情况。而各种交换方法中以

交换材料 1 和 3 之后绝缘数值的降低最小，如果没有禁忌表，则应该选择这个交换。但是，交换材料 1 和 3 之后各种材料的排列顺序又变为 2—4—7—3—5—6—1，回到了上一次交换之前的状态，搜索陷入循环，无法继续进行。可是禁忌搜索算法中由于使用了禁忌表，交换（1，3）因为处于禁忌表中而不能选择，只能选择其他的交换。其他交换中，交换材料 2 和 4 之后绝缘数值降低最小，因此被选中。交换之后，各种材料的排列顺序为 4—2—7—1—5—6—3。

虽然交换（2，4）比（1，3）使绝缘数值的降低更多，但是能将搜索带入一个全新的状态，继续搜索下去完全可能搜索到更好的排列方法。由于使用了禁忌表，避免了循环搜索，因此禁忌搜索算法不会陷入局部最优解。

表 4.1　第一次交换对绝缘效果的影响

交换的材料	绝缘效果的改善情况
1，3	2
2，3	1
3，4	-1
1，7	-2
1，6	-4
…	…

表 4.2　第二次交换对绝缘效果的影响

交换的材料	绝缘效果的改善情况
1，3	-2
2，4	-4
6，7	-6
4，5	-7
3，5	-9
…	…

由于禁忌表中存放的是移动 s 而不是解 x，某些曾经接受的移动完全可能把解引向新的区域，甚至可能得到优于历史最优解的解。因此，如果完全按照上面的禁忌策略来搜索可能遗漏一些区域，所以提出了一个渴望水平函数 $A(s, x)$ 的概念。如果移动 s 达到了渴望水平，即

$c(s(x)) < A(s, x)$，那么这个移动将不受禁忌表的限制而被接受，这称为“破禁”。这样就可能跳离局部最优解，实现全局搜索。当然，有了禁忌策略和渴望水平，迭代还有可能陷入循环。因此，必要时还需要给出其他改进手段，例如对从一个解到另一个解的移动被禁忌的次数进行记录，对被禁忌次数较多的移动实行一定的惩罚策略，记录这些次数的表称为中期表；如果经过多次迭代仍然不能更新历史最优解，可以重新给出初始解，在一个新的区域中开始搜索，这种记录多个初始解的表称为长期表。当迭代达到一定次数，或者满足其他的一些终止条件时，迭代终止。

相对于普通邻域搜索而言，禁忌搜索算法最大的特点是可以接受劣解，这是避免陷入局部最优的首要条件。相应的，禁忌搜索算法不能以局部最优为停止准则，而是设定最大迭代次数或者给出其他特殊的停止准则。可以说，禁忌策略和渴望水平是禁忌搜索算法的两个最核心思想，而两者又是对立统一的。如果能很好地协调禁忌策略与渴望水平的关系，便能很好地实现全局寻优。

4.2 算法的构成要素

禁忌搜索算法中很多构成要素对搜索的速度与质量至关重要，下面将依次给出介绍，包括编码方式（Encode）、适值函数、解的初始化、移动（Moving）与邻域（Neighborhood）、禁忌表（Tabu List）、选择策略（Selection Strategy）、渴望水平（Aspiration Level）和停止准则（Stopping Rule）等。

4.2.1 编码方法

和第3章所讲的遗传算法一样，使用禁忌搜索算法求解一个问题之前，需要选择一种编码方法。编码就是将实际问题的解用一种便于算法操作的形式来描述，通常采用数学的形式；算法进行过程中或者算法结束之后，还需要通过解码来还原到实际问题的解。

根据问题的具体情况，可以灵活地选择编码方式。例如，上节排序问题中采取了顺序编码，各元素的相邻关系表达了各种绝缘材料的排列顺序。对于背包问题，可以采用**0-1**编码，编码的某一位为**0**表示不选择这件物品，为**1**表示选择这件物品。对于实优化问题，一般可以直接使用实数编码，编码的每一位就是解的相应维的取值。

对于同一个问题，也可能有多种编码方式可供选择。例如分组问题：各不相同的n件物品要分为m组，满足特定的约束条件，要达到

特定的目标函数。以下两种编码方式都是可行的。

1. 编码 1

以自然数 $1\sim n$ 分别代表 n 件物品，n 个数加上 $m-1$ 个分割符号（例如用 **0** 表示）混编在一起，随意排列，便得到一种编码方式。例如，$n=9$，$m=3$，下面便是一个合法的编码：

1—3—4—**0**—2—6—7—5—**0**—8—9

这种可以称为带分隔符的顺序编码。

2. 编码 2

编码的每一位分别代表一件物品，而每一位的值代表该物品所在的分组。同样是 $n=9$，$m=3$ 的情况，可以给出如下形式的编码：

1—2—1—1—2—2—2—3—3

这种是一般的自然数编码。

不同编码形式通常是可以相互转化的，例如带分隔符的顺序编码与一般的自然数编码就很容易相互转化。事实上，上面给出的两个编码表示的是同一个解，也就是物品 1、3、4 分在第一组；物品 2、6、7、5 分在第二组；其余的物品 8、9 分在第三组。

注意到：如果稍微修改编码 1 中给出的编码举例 1—3—4—**0**—2—6—7—5—**0**—8—9，交换元素 1 和 3 的位置，得到如下一个新的编码：3—1—4—**0**—2—6—7—5—**0**—8—9。这两个编码是不同的，但是对应的解是相同的。对于编码 2，如果各个组是没有区别的，3—2—3—3—2—2—2—1—1 对应的解和原来编码对应的解也是一样的，只不过是将组的标号做了调换。这种多个编码对应同一个解，即编码空间的大小大于解空间的大小的情况是不希望出现的。实际应用中，希望编码空间尽可能和解空间一样大小，也就是说编码和解具有严格的一一对应关系。然而，对于许多的实际问题，这并不是一件容易的事情。

4.2.2 适值函数的构造

类似于遗传算法，适值函数也是用来对搜索状态进行评价的。将目标函数直接作为适值函数是最直接也是最容易理解的做法。当然，对目标函数的任何变形都可以作为适值函数，只要这个变形是严格单调的。例如，式（4.1）中目标函数为 $c(x)$，设适值函数用 $c'(x)$ 表示，那么

$$c'(x)=\begin{cases} kc(x)+b & k\neq 0 \\ (c(x))^2 & c(x)>0 \\ a^{c(x)} & a>0, a\neq 1 \\ \cdots & \end{cases}$$

都是可以的，只要在选择的时候注意一下这个变形应该和原来目标函数的大小顺序保持一致。这和遗传算法的适值函数的标定是同一道理。

适值函数的选择主要考虑提高算法的效率、便于搜索的进行等因素，以上给出的各种变形都是针对特定的目标函数形式为了简化算法而设计的。例如，某些问题中目标函数为多个偏差的方均根，即 $c(x)=\sqrt{\dfrac{\sum_{i=1}^{n}e_i^2}{n}}$，其中 $e_i(i=1,\cdots,n)$ 为 n 个偏差，那么适值函数可以取为偏差的平方和，即 $c'(x)=n(c(x))^2=\sum_{i=1}^{n}e_i^2$，因为这样的适值函数与目标函数具有同样的增减性，最小化 $c'(x)$ 的同时也最小化了 $c(x)$，同时避免了不必要的除法与开方运算。再如，某些工业过程的目标函数需要一次仿真才能得到，如果选择其中一些反映目标函数的特征参数作为适值函数，可以大大节省计算时间。

4.2.3 初始解的获得

禁忌搜索算法可以随机给出初始解，也可以事先使用其他启发式等算法给出一个较好的初始解。由于禁忌搜索算法主要是基于邻域搜索的，初始解的好坏对搜索的性能影响很大。尤其是一些带有很复杂约束的优化问题，如果随机给出初始解很可能是不可行的，甚至通过多步搜索也很难找到一个可行解，这个时候应该针对特定的复杂约束，采用启发式方法或其他方法找出一个可行解作为初始解。

4.2.4 移动与邻域移动

移动是从当前解产生新解的途径，例如问题（P）中用移动 s 产生新解 $s(x)$。从当前解可以进行的所有移动构成邻域，也可以理解为从当前解经过“一步”可以到达的区域。适当的移动规则的设计，是取得高效的搜索算法的关键。

邻域移动的方法很多，求解不同的问题需要设计不同的移动规则。例如，前面排序问题中采用两两交换式的移动规则，而背包问题中可能采用修改解中任意一个元素的值的移动规则。而在另外一些问题中，移动可能被定义为一系列复杂的操作。禁忌搜索算法中的邻域移动规则和遗传算法中的交叉算子和变异算子相似，需要根据特定的问题来设计，在此不一一列举。

4.2.5 禁忌表

在禁忌搜索算法中，禁忌表是用来防止搜索过程中出现循环，避免陷入局部最优的。它通常记录最近接受的若干次移动，在一定次数之内禁止再次被访问；过了一定次数之后，这些移动从禁忌表中退出，又可以重新被访问。禁忌表是禁忌搜索算法中的核心，它的功能和人类的短期记忆功能十分相似，因此又称为“短期表”。

1. 禁忌对象

所谓禁忌对象就是放入禁忌表中的那些元素，而禁忌的目的就是避免迂回搜索，尽量搜索一些有效的途径。禁忌对象的选择十分灵活，可以是最近访问过的点、状态、状态的变化以及目标值等。例如上述7元素排序问题中，把两两交换的对象作为禁忌对象，也就是禁忌了状态的变化。有些问题中可以将状态本身作为禁忌对象，例如：背包问题中选择“取”或者“不取”某物品为禁忌对象；分组问题中把某元素和它被分到的组联系在一起，构成一个有序数对作为禁忌对象；还有些问题中可以把目标值直接放入禁忌表中作为禁忌对象。

尽管可以有多种方式给出禁忌对象，但是归纳起来，主要有如下三种：

(1) 以状态的本身或者状态的变化作为禁忌对象。例如把移动 s 或者从当前解到新解的改变 $x \to s(x)$ 放入禁忌表中，禁止以后再做这样的移动，避免搜索循环。选择这种禁忌对象比较容易理解，但是禁忌的范围比较小，只有和这些完全相同的状态才被禁忌，搜索空间很大。而存储禁忌对象所占的空间和所用的时间却比较多。

(2) 以状态分量或者状态分量的变化作为禁忌对象。对于一些维数很大的问题，每一次移动可能都有很多状态分量发生变化。如果只取其中一个分量或者少数几个分量作为禁忌对象，那么禁忌的范围比较大，而且存储的空间和时间都比较少。例如移动 $s=(s_1, s_2, \cdots, s_d)$，其中 d 为这种移动包含的分量个数，可以取 $s_i(1 \leqslant i \leqslant d)$ 作为禁忌对象。

(3) 采取类似于等高线的做法，将目标值作为禁忌对象。这种做法将具有相同目标值的状态视为同一个状态，大大增加了禁忌的范围。对于上述例子，可以采用 $c(s(x))$ 作为禁忌对象。

这三种做法中，第一种做法的禁忌范围适中，第二种做法的禁忌范围较小，第三种做法的禁忌范围较大。如果禁忌范围比较大，则可能陷入局部最优解；反之，则容易陷入循环。实际问题中，要根据问题的规

模、禁忌表的长度等具体情况来确定禁忌对象。

2. 禁忌长度

所谓禁忌长度（Tabu Size），就是禁忌表的大小。一个禁忌对象进入禁忌表后，只有经过一个确定的迭代次数，才能从禁忌表中退出。也就是说，在当前迭代之后的确定次迭代中，这个发生不久的相同操作是被禁止的。容易知道，禁忌表的长度越小，计算时间和存储空间越少，这是任何一个算法都希望的；但是，如果禁忌长度过小，会造成搜索的循环，这又是要避免的。

禁忌长度不但影响了搜索的时间，还直接关系着搜索的两个关键策略：局域搜索策略和广域搜索策略。如果禁忌表比较长，便于在更广阔的区域搜索，广域搜索性能比较好；而禁忌表比较短，则使得搜索在小的范围进行，局域搜索性能比较好。禁忌长度的设定要依据问题的规模、邻域的大小来确定，从而达到平衡这两种搜索策略的目的。

总结起来，主要有如下一些设定禁忌长度的方法。

①禁忌长度 t 固定不变。t 可以取一些与问题无关的常数，例如 $t=5$，7，11 等值。也有一些有学者认为：禁忌长度应该与问题的规模有关，例如取 $t=\sqrt{n}$，这里 n 为问题的规模。这种方法方便简单，而且容易实现。

②禁忌长度 t 随迭代的进行而改变。根据迭代的具体情况，按照某种规则，禁忌长度在区间$[t_{min}, t_{max}]$内变化。这个禁忌长度的区间可以与问题无关，例如 [1，10]；或者与要求解问题的规模有关，例如$[0.9\sqrt{n}, 1.1\sqrt{n}]$。而这个区间的两个端点也可以随着迭代的进行而改变。

大量研究表明，动态的设定禁忌长度比固定不变的禁忌长度具有更好的性能。关于禁忌长度的更深入探讨，见本章关于主动禁忌搜索算法的介绍。

4.2.6 选择策略

选择策略就是从邻域中选择一个比较好的解作为下一次迭代初始解的方法，用公式可以表示为

$$x' = \underset{s(x)\in V}{\mathrm{opt}}\, s(x) = \arg[\underset{s(x)\in V}{\max/\min}\, c'(s(x))] \tag{4.2}$$

其中，x 为当前解，x'为选出的邻域最好解，$s(x)\in V$ 为邻域解，$c'(s(x))$ 为候选解 $s(x)$ 的适值函数，$V\subseteq S(x)$ 称为候选解集，它是邻域的一个子集。要根据问题的性质和适值函数的形式，在候选解集中选择一个最

好的解。然而，候选解集的确定，与上文讨论的禁忌长度的大小相似，对搜索速度与性能影响都很大。

（1）候选解集为整个邻域，即 $V=S(x)$。这种选择策略就是从整个邻域中选择一个最优的解作为下一次迭代的初始解。这种策略择优效果好，相当于选择了最速下降方向，但是要扫描整个邻域，计算时间比较长，尤其对于大规模的问题，这种策略可能让人无法接受。在上文的7元素示例中，采用的就是这种方法，只是限于篇幅，只列出了比较好的前几个移动，小规模问题中这种策略是可以的。

（2）候选解集为邻域的真子集，即 $V\subset S(x)$。这种策略只扫描邻域的一部分来构成候选解集，甚至是一小部分，$|V|\ll|S(x)|$，这里 $|V|$ 和 $|S(x)|$ 分别表示候选解集和邻域的大小。这种策略虽然不一定取到了邻域中的最好解，但是节省了大量的时间，可以进行更多次迭代，也可以找到很好的解。极限情况下，可以选择第一个找到的改进解，也就是说：只要发现了改进解，马上停止扫描。当然，如果整个邻域中没有改进解，那么只好选择一个最好的劣解了。

注：本节讨论选择策略的过程中没有考虑禁忌表。实际上，其中的邻域应该是邻域中除了禁忌解之外的区域，可以表示为 $S(x)-T$。

禁忌搜索算法在每一步迭代的过程中，都包含了启发式思想，是启发式的启发式，正是从这个角度才被称为元启发式算法（Meta-Heuristics）。

4.2.7 渴望水平

在某些特定的条件下，不管某个移动是否在禁忌表中，都接受这个移动，并更新当前解和历史最优解。这个移动满足的这个特定条件，称为渴望水平（Aspiration Level），或称为破禁水平、特赦准则、蔑视准则等，例如下述7元素排序问题示例中，当迭代到第4步的时候，移动（4，5）虽然在禁忌表中，仍旧选择了它，这是因为这个移动能得到一个超过历史最优解的解，能使历史最优值从18变为20，满足了渴望水平。

渴望水平的设定也有多种形式，总结起来如下。

（1）基于适配值的准则。如果某个候选解的适配值优于历史最优值，也称为“Best so Far”状态，那么无论这个候选解是否处于被禁忌状态，都会被接受。对于开始提到的问题（P），这种渴望水平可以描述为

$$c(s(x))<c(x^*) \tag{4.3}$$

这个准则最容易理解，应用得也最广泛，例如下述7元素排序问题的示例中应用的就是这个准则。直观上理解，这个准则就是找到了更好的

解，那么即使这个移动被禁忌了，也要给它破禁，同时要更新历史最优解。

然而，如果只选用这一种渴望水平，那么可能错过一些有潜力的区域。有一些移动虽然不能立即带来优于历史最优解的解，但是有这个潜力，可能在接下来的几步迭代中超过历史最优，于是出现了其他渴望水平。

（2）基于搜索方向的准则。如果某禁忌对象进入禁忌表的时候改善了适配值，而这次这个被禁忌的候选解又改善了适配值，那么这个移动破禁。对于问题（P），这个准则可以描述为

$$c(s(x)) < c(x) \text{ 且 } c(s(\underline{x})) < c(\underline{x}) \tag{4.4}$$

其中$\underline{x}$表示该对象上次被禁忌时的解。这种准则也可以很好地避免搜索循环，一般的，如果这次经过某个状态时改善了适配值，下一次经过这个状态时恶化了适配值，那么很可能是按原路返回了。例如上文7元素排序问题中，第二步迭代交换1和3适配值增加2，而紧接着，如果再交换1和3，适配值就减少2，明显是回到了原来的状态，尽管这是当前最好的移动，还是不能接受，避免了搜索循环。直观上看，这种策略可以理解为算法正在按有效的方向继续搜索。

（3）基于影响力的准则。迭代过程中，不同对象对适配值的影响不同，有的较大，有的较小。影响较大的可能是问题的主要因素，影响较小的可能是次要因素。可以结合禁忌对象在禁忌表中的位置，即进入禁忌表时间的长短和对适配值影响力的大小，来制定渴望水平。对于这种策略的直观理解为：解禁一个影响力较大的对象，可能对适配值有较大的影响。这里影响力可以是增加适配值，也可以是减少适配值。当然，这种做法需要额外地制定一个衡量一个对象影响力大小的方案，增加了算法的复杂性。

（4）其他准则。例如，当所有候选解都被禁忌，而且不满足上述渴望水平，那么可以选择其中一个最好解来解除禁忌，否则，算法将无法继续下去。事实上，这种所有候选解都被禁忌而且不优于历史最优解的情况是应该尽量避免的，比如可以设置禁忌长度小于邻域的大小。

渴望水平的设计比较灵活，实际应用中可以采取上述准则中的一种，或者同时选择其中的几种。而且，渴望水平还要与禁忌长度、候选解集等策略综合考虑，平衡集中强化搜索与分散多样化搜索。

禁忌策略与渴望水平是禁忌搜索算法中两大核心策略，两者是对立统一的，也可以看做是一个统一原则的两个方面。可以通过设置不同的禁忌长度来禁止一些对象，又可以通过设置不同的渴望水平来解除一些

对象，从而实现全局搜索。

4.2.8 停止准则

和其他元启发式算法，包括遗传算法、模拟退化算法等一样，禁忌搜索算法不能保证找到问题的全局最优解，而且没有判断是否找到全局最优解的准则。因此，必须另外给出停止准则来停止搜索，常用的包括如下几种。

①给定最大迭代步数。这个方法简单容易操作，在实际中应用最为广泛。

②得到满意解。如果事先知道问题的最优解，而算法已经达到最优解，或者与最优解的偏差达到满意的程度，则停止算法。这种情况常应用于算法效果的验证，因为只有这个时候问题的最优解才可能是事先知道的。或者在实际应用中，用其他估界算法已经估算出问题的上界（目标函数是最大化）或者下界（目标函数是最小化），如果搜索得到的历史最优值与这些“界”的偏差满足要求，停止算法，其实这也是得到了满意解。

③设定某对象的最大禁忌频率。如果某对象的禁忌频率达到了事先给定的阈值，或者历史最优值连续若干步迭代得不到改善，则算法停止。

4.3 算法流程与算例

以上给出了用禁忌搜索算法求解排序问题的一个简单示例，而后介绍了禁忌搜索算法中的一些基本概念。简单地讲，禁忌搜索算法以禁忌表来记录最近搜索过的一些状态，对于当前邻域中一个比较好的解，如果不在禁忌表中，那么选择它作为下一步迭代的初始解，否则宁愿选择一个比较差的但是不在禁忌表中的解；而如果某个解或者状态足够得好，则不论其是否在禁忌表中，都接受这个解；如此迭代，直至满足事先设定的停止准则。

4.3.1 基本步骤

由于禁忌搜索算法的渴望水平、选择策略以及停止准则等都可以有多种设定方式，如果再使用中期表和长期表，禁忌搜索算法的步骤将多种多样。下面给出一个最基本的步骤（不考虑中期表和长期表）。

第 1 步：初始化。给出初始解，禁忌表设为空。

第 2 步：判断是否满足停止条件。如果满足，输出结果，算法停

止；否则继续以下步骤。

第3步：对于候选解集中的最好解，判断其是否满足渴望水平。如果满足，更新渴望水平，更新当前解，转第5步；否则继续以下步骤。

第4步：选择候选解集中不被禁忌（不对应于禁忌表中的一个对象）的最好解作为当前解。

第5步：更新禁忌表。

第6步：转第2步。

当然，这样的步骤也不可能概括禁忌搜索算法的各种情况，只是给出一个一般的描述供读者参考。

4.3.2 流程图

以本章开始给出的问题（P）为例，如果以整个邻域为候选解集，以目标函数作为适值函数，以给定最大代数 NG 为停止准则，以优于历史最优解为渴望水平，则算法的流程如图4.1所示。

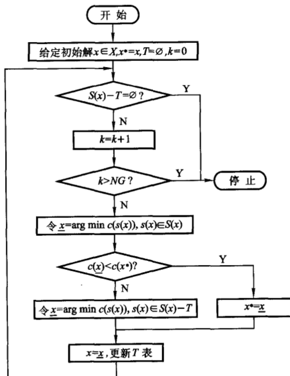


图4.1 禁忌搜索算法的流程图1

从图 4.1 中可以看到，这种方法中当所有邻域解都被禁忌时，算法异常终止了。这是禁忌长度过大、而邻域过小造成的，设计算法时一般应该避免这种情况的发生。如果以第一个改进解为候选解，当所有邻域解都被禁忌时，选择其中最好的解来破禁，当然发现优于历史最优解时则不考虑禁忌状态，可以得到另外一种流程图，如图 4.2 所示。

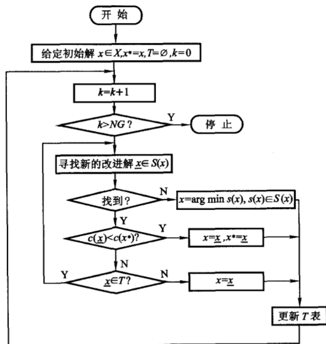


图 4.2 禁忌搜索算法的流程图 2

可见，如果禁忌搜索算法中的各环节采取的策略不同，得到的程序流程图也是不同的，究竟采取哪种策略，应该根据问题的具体情况确定。

同传统的优化算法相比，禁忌搜索算法具有如下优点：

①能接受劣解，具有很好的爬山能力。

②区域集中搜索与全局分散搜索能较好平衡。

但是，以上介绍的基本禁忌搜索也有明显的不足：

①对初始解和邻域结构有较大的依赖性，一个好的初始解可能很快迭代到最优解，一个较差的初始解可能会极大地降低搜索质量。

②搜索过程是串行的，不像遗传算法那样具有并行的搜索机制。

为了全面提高禁忌搜索算法的性能，可以针对其中的关键策略以及参数设置等方面进行改进，也可以与其他优化算法相结合。值得指出的是，这些改进后的算法（或者混合算法）仍然可以叫做“禁忌搜索算法”，因为禁忌搜索算法并没有规定必须是上述基本禁忌搜索算法的情况，禁忌搜索算法是使用“禁忌机制”和“赦免准则”引导搜索的思想。与其说禁忌搜索算法是一种方法，不如说是一种技术，或者说是一门艺术。只有应用得好，才能很好地解决实际问题。

4.3.3 一个简单的例子

下面以一个7元素的排序问题为例，来说明禁忌搜索算法的算法流程。问题的背景可以理解为：由七种不同的绝缘材料构成一种绝缘体，如何排列才能使得绝缘的效果最好？

组合优化问题是禁忌搜索算法应用得最广泛的领域，而大量的组合优化问题中，排序问题是典型代表，如旅行商问题（TSP）、作业调度问题等。对于一个n元素的排序问题，所有解的数目即解空间的大小为n的全排列$P_n^n = n!$。当n较大时，这将是一个天文数字，穷举搜索是不可能完成的。使用禁忌搜索算法可以在较少的迭代后得到一个满意解。

首先确定编码方式，这里采用顺序编码，即1~7共7个数字的一个排列便是一种合法的编码。定义互换操作为这个问题的邻域结构，任意交换两种材料的位置，便得到一个邻域解。这样，对于每一个解，它的邻域解的个数为$C_7^2 = 21$个，从当前解到邻域中的另外一个解的改变称为一次邻域移动。然后，设计禁忌表的结构：以互换的两种材料（以1~7的编码表示）构成的数对作为禁忌表的元素。禁忌表的长度取为3，也就是说：当第4个元素进入禁忌表时，第1个元素从禁忌表中退出。以绝缘效果作为目标函数值，目标函数值越大越好。最后给出渴望水平：如果当前解的某移动得到的解优于历史最优解，则不论该移动是否在禁忌表中，都将接受作为下一次迭代的初始解。设定最大迭代（移动）次数为停止准则，本例中最大迭代次数取为5。

初始状态：随机给出一个初始解为：2—5—7—3—4—6—1，目标函数值为10，历史最优值也为10，禁忌表为空。由给定解计算出目标函数值的具体过程与本算法关系不大，在此不详细介绍，而是直接给出目标函数值的改变情况。交换解的任意两个元素，得到新解的目标函数值与当前目标值之差。限于篇幅，这里只列出最好的5个移动。这个状态如图4.3所示。

移动 $s(x)$	适配值改变 $\Delta c(x)$
4，5	6
4，7	4
3，6	2
2，3	0
1，4	-1
…	…

当前解 x：2—5—7—3—4—6—1；
历史最优值 $c(x^*)=10$；当前值 $c(x)=10$。

禁忌表	
1	∅
2	∅
3	∅

图 4.3 7 元素排序问题初始状态

第 1 步：当前邻域中目标函数值改善最大的移动是（4，5），即元素 4 和 5 交换，可以增加目标值 6，而这个移动不在禁忌表中，所以本次迭代中选择了这个移动。当前解变为 2—4—7—3—5—6—1；目标函数值 $c(x)=10+6=16$；历史最优值 $c(x^*)=16$；将移动（4，5）加入禁忌表中。针对当前解，交换解的任意两个元素，得到目标函数值改善最大的 5 个移动如图 4.4 所示。

移动 $s(x)$	适配值改变 $\Delta c(x)$
1，3	2
2，3	1
3，4	-1
1，7	-2
1，6	-4
…	…

当前解 x：2—4—7—3—5—6—1；
历史最优值 $c(x^*)=16$；当前值 $c(x)=16$。

禁忌表	
1	4，5
2	∅
3	∅

图 4.4 7 元素排序问题第一次迭代后状态

第 2 步：当前邻域中目标函数值改善最大的移动是（1，3），而且不在禁忌表中，故本次迭代选择这个移动。当前解变为 2—4—7—1—5—6—3；目标函数值 $c(x)=18$；历史最优值 $c(x^*)=18$；将移动（1，3）加入禁忌表中，同时禁忌表中的原有元素下移一个位置。针对当前解，交换解的任意两个元素，得到目标函数值的改善情况，这个状态如图 4.5 所示。

移动 $s(x)$	适配值改变 $\Delta c(x)$
1，3	-2
2，4	-4
6，7	-6
4，5	-7
3，5	-9
…	…

当前解 x：2—4—7—1—5—6—3；
历史最优值 $c(x^*)=18$；当前值 $c(x)=18$。

禁忌表	
1	1，3
2	4，5
3	∅

图 4.5 7元素排序问题第二次迭代后状态

第3步：当前邻域中所有移动都不能改善当前解，证明当前解就是一个局部最优解，如果按照普通的邻域搜索规则，算法就停止了。可是禁忌搜索算法接受劣解，所以算法才能继续。目标函数值改善最大的（尽管是劣解，为了统一，仍然使用这个说法）移动为（1，3）。但是，这个移动已经在禁忌表中，而且 $c(x)-2<c(x^*)$，说明这个移动得到的解不能改善历史最优解，没有达到渴望水平。所以选择次优的移动（2，4），当前解变为 4—2—7—1—5—6—3；当前值为 $c(x)=14$，而历史最优值不变，修改禁忌表。再一次考察所有移动得到的目标函数值改变情况，相应状态如图 4.6 所示。

移动 $s(x)$	适配值改变 $\Delta c(x)$
4，5	6
3，5	2
1，7	0
1，3	-3
2，6	-6
…	…

当前解 x：4—2—7—1—5—6—3；
历史最优值 $c(x^*)=18$；当前值 $c(x)=14$。

禁忌表	
1	2，4
2	1，3
3	4，5

图 4.6 7元素排序问题第三次迭代后状态

第4步：邻域中最优移动为（4，5），该移动能使目标值增加6，则目标函数值变为 $c(x)+6>c(x^*)$，优于历史最优值，达到了渴望水平，虽然这个移动在禁忌表中，仍然接受。当前解变为 5—2—7—1—4—6—3；当前值 $c(x)=20$；相应的，历史最优值 $c(x^*)=20$，修改禁忌表。考察所有移动对应目标函数值的改善情况，得到状态如图 4.7 所示。

移动 $s(x)$	适配值改变 $\Delta c(x)$
1, 7	0
3, 4	-3
3, 6	-5
4, 5	-6
2, 6	-8
…	…

当前解 x：5—2—7—1—4—6—3；
历史最优值 $c(x^*)=20$；当前值 $c(x)=20$。

禁忌表	
1	4, 5
2	2, 4
3	1, 3

图 4.7　7 元素排序问题第四次迭代后状态

第 5 步：邻域中最优移动为（1，7），不在禁忌表中，所以接受。当前解变为 5—2—1—7—4—6—3；最优值不变。由于已经迭代 5 次，达到了预先设定的最大迭代次数，算法停止。最优目标值为 20，由于这次迭代没有改变目标函数值，所以得到两个等价的最优解 5—2—1—7—4—6—3 和 5—2—7—1—4—6—3。

该算例说明了禁忌搜索算法的基本思想：禁忌策略与渴望水平，描述了简单禁忌搜索算法的步骤。注意到，禁忌搜索算法中包括上一节中介绍的编码、初始化、邻域结构设计、选择策略、禁忌表、渴望水平、停止准则等概念。

4.4　中期表与长期表

4.2 节给出的是禁忌搜索算法中的一些基本概念，有了这些概念，就可以使用禁忌搜索算法来求解问题了。禁忌搜索算法的局域选优能力很好，邻域选优速度快，但是广域搜索能力较差。而且，仅依靠记录状态、状态变化、状态分量变化或者适配值变化等对象的禁忌表，也就是短期表，禁忌搜索算法不能避免较大的搜索循环。为此，禁忌搜索算法中又引入了中期表和长期表的概念。

与短期表一起，中期表和长期表在“学习式”搜索和“非学习式”搜索之间起到一个交互式作用。使用中期表和长期表，可以达到在区域内强化搜索和全局多样化搜索的效果，下面分别给出介绍。

4.4.1　中期表

中期表也称为频数表或频率表。禁忌频数（频率）是对禁忌属性的一种补充，可以放宽选择决策对象的范围。例如，如果某个适配值频

繁出现，可以推测搜索陷入了循环或者达到了某个极值点，或者说算法当前参数很难搜索到更好的状态，需要调整参数，以期望更好的效果。

实际应用中，可以根据问题和算法的需要，记录某些状态出现的频率，某些状态变化或者适配值的变化信息，而这些信息可以是静态的，也可以是动态的。

1. 静态信息

可以记录搜索过程中某些交换、状态变化或者某些适配值出现的频数、频率（某对象出现的频数与总迭代次数的比）等信息。对那些频繁出现的对象进行惩罚，使算法进行更为有效的搜索。

以本章开始时给出的问题（P）为例，在不考虑中期表的情况下，每一次迭代是在候选解集中选择一个目标函数值最小的解，即

$$\min c(s(x)),\ s(x)\in V\subset S(x)-T \tag{4.5}$$

如果考虑了中期表，那么每一次迭代是在候选解集中选择目标函数值小而且被禁忌次数比较少的解，即

$$\min c(s(x))+\alpha N(s(x)),\ s(x)\in V\subset S(x)-T \tag{4.6}$$

其中，$N(s(x))$为$s(x)$曾经被禁忌的频数，α是惩罚因子。惩罚因子α的取值应该远小于目标函数值，一般取目标函数值的1% ~1‰。惩罚因子的取值用来平衡中期表和短期表之间的效果，α取值越大，分散的效果越好，但是会破坏邻域搜索的性能。

中期表的记录方法也有一些技巧。以n元素排序问题为例，可以将短期表与中期表放在一个矩阵中。建立一个$n\times n$的矩阵，其中右上部分作为短期表，左下部分作为中期表，对角线上元素没有意义。每一次迭代，将短期表中大于1的元素减1（减至0则该元素退出短期表），新加入短期表的元素置为禁忌长度，这样短期表中始终有禁忌长度个元素；而每一次迭代，中期表中新加入的元素加1，记录该元素被禁忌的次数。对于以上7元素排序问题示例，迭代3次和4次后的短期表和中期表用这种矩阵表示分别为

$$\begin{bmatrix} - & & 2 & & & & \\ & - & & 3 & & & \\ 1 & & - & & & & \\ & 1 & & - & 1 & & \\ & & & 1 & - & & \\ & & & & & - & \\ & & & & & & - \end{bmatrix} \text{和} \begin{bmatrix} - & & 1 & & & & \\ & - & & 2 & & & \\ 1 & & - & & & & \\ & 1 & & - & 3 & & \\ & & & 2 & - & & \\ & & & & & - & \\ & & & & & & - \end{bmatrix}$$

矩阵中没有列出的元素为0。这里主要为了说明这种表示方法，没有考虑引入中期表后对搜索结果的影响。

这种记录方法，短期表和中期表共用一个矩阵，既方便操作又节省了存储空间。当然，当问题规模很大时，矩阵中为零的元素个数大大增加，这个矩阵成为稀疏矩阵，这种记录方法不一定很好，中期表和短期表可能需要分别记录。

2. 动态信息

主要记录从某些状态或者适配值等对象转移到另一些状态或者适配值等对象的变化趋势，例如记录某些序列的变化。显然这种中期表需要记录的内容较多，而提供的信息量也较大。常用的有如下几种记录方法。

①记录某个序列的长度。

②记录由某个元素出发再回到这个元素需要的迭代次数。

③记录序列中适配值的平均值，或者序列中各元素的适配值。

④记录某个序列出现的频率等。

记录这些信息之后，可以对某些序列进行惩罚，或者采用更复杂的处理方式。

中期表和短期表都是基于已经经历过的搜索给出的策略，属于“学习式”搜索。

4.4.2 长期表

使用短期表的搜索方式可以认为是邻域搜索，而使用中期表的方式可以认为是区域强化式（Regional Intensification）搜索，但仍可能达不到全局搜索。为了实现全局多样化（Global Diversification）搜索，提出了长期表的概念。

长期表用来记录多个初始解，从这些初始解开始分别进行禁忌搜索，即多阶段禁忌搜索。产生一个初始解时，应该尽量与已经产生的初始解保持较远的距离，使得各个初始解在可行域内具有良好的分散性，以更好地在全局进行搜索。

对于一个 n 维问题，其中第 k 个初始解 $x^k=(x_1^k,\cdots,x_n^k)$ 可以取

$$x^k = \arg\max D^k = \arg\max \sum_{l\in L}\sum_{i=1}^{n}(x_i^k - x_i^l)^2 \tag{4.7}$$

其中，x_i^l，$l\in L$ 为已经选定的初始解的第 i 个元素，即 $x^l=(x_1^l,\cdots,x_n^l)$。

不同于短期表和中期表，这种长期表不是基于过去的搜索进行搜索，而是在新的区域内完全随机生成初始解进行搜索，属于“非学习

式”搜索策略。使用长期表进行多阶段禁忌搜索，能很好地提高算法的广域搜索能力，同时不丧失禁忌搜索算法的邻域搜索能力。

多阶段禁忌搜索是后来发展的多种并行禁忌搜索中的一种，关于并行禁忌搜索，将在后文详细介绍。

4.5 算法性能的改进

禁忌搜索算法具有全局寻优能力，而且比较容易实现，自从20世纪90年代就引起了广泛的重视。但是应用中也发现，以上基本的禁忌搜索算法有一些缺点，对于给定的实际工程问题，可能需要大量的调试工作才能得到较好的效果，于是提出一些改进做法。下面介绍几种比较主要的改进，包括并行禁忌搜索算法、主动禁忌搜索算法以及禁忌搜索算法和其他算法的混合策略等。

4.5.1 并行禁忌搜索算法

随着并行计算技术和并行计算机的发展，为满足求解大规模优化问题的需要，禁忌搜索算法的并行实施也得到了研究与进展。

相对于前面介绍的基本禁忌搜索算法，对算法的初始化、参数设置、通信方式等方面实施并行策略，能得到各种不同类型的并行禁忌搜索算法。当前，对并行禁忌搜索算法比较认可的一种分类方法如图4.8所示。

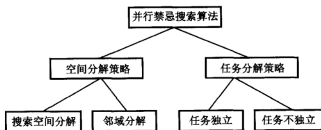


图4.8 并行禁忌搜索算法分类

1. 基于空间分解策略

基于空间的分解策略包括搜索空间分解和邻域分解两种做法。

搜索空间分解，即通过搜索空间的分解将原问题分解为多个子问题分别进行求解，从而实现并行化。这里求解各个子问题的算法参数可以相同，也可以不同。

前文介绍禁忌搜索算法中基本概念“长期表”时，提到过多阶段

禁忌搜索算法。多阶段禁忌搜索算法中，首先产生多个彼此距离比较远的初始解，记录在长期表中，然后从这些初始解出发，分多个阶段对整个问题进行搜索。如果从这些初始解出发，同时进行搜索，就是这里讲的一种并行禁忌搜索算法。

邻域分解策略，即每一步中用多种方法对邻域分解得到的子集进行评价，从而实现对最佳邻域解搜索的并行化，这种分解策略对同步的要求比较高。

2. 基于任务分解策略

将待求解问题分解为多个任务，每一个任务使用一个禁忌搜索算法来求解。不同的禁忌搜索算法可以设置不同的参数，包括初始解、邻域结构、选择策略、渴望水平等。在多处理机的情况下，根据这些任务对各处理机的分配情况，又可以分为如下三种。

①非自适应方式。任务的数量和定位在编译时就已经确定，各任务在各处理机中的定位在算法进行过程中是不变的，也就是静态的调度方式。例如，根据处理机的个数将搜索空间分解为一些子空间分别进行搜索。这种方法中，很容易造成各处理机之间任务不平衡的情况，因而造成有些处理机长期处于空闲状态，影响搜索的整体效率。但是，由于实现起来比较容易，当前大多数并行禁忌搜索算法采用的都是这种方式。

②半自适应方式。任务的数量在编译时给定，而各任务的定位在运行时给定。这种方式相对于非自适应方式有了一定的改进，运行时可以在一定程度上平衡各处理机的负荷，但仍不能实现彻底的平衡。

③自适应方式。任务的生成和分配完全是动态的，是在运行时给出的。当某处理机空闲时，则生成新的任务；当处理机繁忙时，则取消某任务。Talbi 等（1998）提出了一种并行自适应禁忌搜索算法，算法由并行而独立的子禁忌搜索算法构成，各算法的各种运行参数独立给出而且可以不同，各任务之间没有通信，并通过二次指派问题（QAP）的高效求解验证了算法的有效性。

空间分解策略有较强的问题依赖性，只对某些问题适用，而基于任务分解策略具有较高的适用性。当然也可以结合空间分解策略和任务分解策略，设计混合的并行策略来求解问题。

董宏光等（2004）将并行禁忌搜索算法引入到化工行业的精馏分离序列综合问题中。精馏分离序列综合问题是一种混合整数非线性规划问题，与组合数学中的一些著名问题，如凸多边形三角划分、矩阵链乘等具有相同的本质特征。由于采用二叉树模式简捷地描述了可行分离序列，算法采用数字串形式编码。算法以相对费用函数为评价指标（适

值函数)，当某候选解优于历史最优解时，无视其禁忌属性直接选为下一代初始解；当所有候选解都被禁忌时，选择最好的候选解破禁。当搜索达到最大给定代数，或者在给定代数内最优值没有改进时，算法终止。最后给出的10组分精馏分离序列算例表明，即使随机给出初始解，寻优结果通常也是全局最优的。

4.5.2 主动禁忌搜索算法

1. 基本禁忌搜索算法的困惑

基本禁忌搜索算法相对于传统的优化方法而言，具有很好的爬山能力，能够避免陷入局部最优点。相对于遗传算法等其他优化方法而言，禁忌搜索算法计算速度比较快，因而得到广泛的应用。但是，对于前面介绍的基本禁忌搜索算法，研究人员遇到了一些困惑。

(1) 参数调整比较困难。和其他元启发式算法包括遗传算法、模拟退火算法等相似，禁忌搜索算法需要设置或者调整一些参数来进行有效的搜索。然而要得到合适的参数，不仅依赖于待求解的具体问题，而且相当费时。因此，参数调整的困难是各种元启发式算法需要解决的一个突出问题。

基本禁忌搜索算法中，候选解集的大小需要调整，禁忌长度需要设定。仅以禁忌长度为例，尽管已经做了大量的研究工作，从起初的与问题无关的固定常数，如5、7、11等；到依赖于问题规模 n 的常数，如$\sqrt{n}$；到给出禁忌长度的两个极限，形如[1, 10]，或者$[0.9\sqrt{n}, 1.1\sqrt{n}]$等，没有哪一种方法能适合于所有问题，而且没有给出设定这些参数的理论依据。面对一个给定的实际问题，常常是经过各种尝试，最后才得到一种可以接受的方案。

(2) 不能避免循环。禁忌表的提出就是为了尽量避免迂回搜索，而禁忌表也确实在很大程度上避免了循环。但是，禁忌搜索算法不能避免较大的循环。即使在引入了中期表和长期表之后，也不能彻底地避免循环。当局部最优点的周围被一些大的“吸引盆”包围的时候，禁忌搜索算法收敛得相当慢。

当禁忌搜索算法得到一个局部最优点时，使用禁忌表禁止刚访问过的点，使得搜索逐渐远离局部最优点。这里有一个隐含的假设：从局部最优点出发，而不是从随机点出发，能更容易地达到全局最优点。但是，研究表明有时候可能不是这样。

2. 主动禁忌搜索算法的基本原理

主动搜索 (Reactive Search，简称 RS) 是一种反馈机制，是一种适

合于求解离散优化问题的启发式算法。Battiti 和 Tecchiolli（1994）将主动搜索机制引入到禁忌搜索算法中来，提出了主动禁忌搜索（Reactive Tabu Search，简称 RTS）算法。

主动禁忌搜索算法利用反馈机制自动调整禁忌表长度，自动平衡集中强化搜索策略和分散多样化搜索策略。算法中给出增大调节系数 N_{IN}（$N_{IN}>1$）和减小调节系数 N_{DE}（$0<N_{DE}<1$）。搜索过程中，所有访问过的解都被存储起来，每当执行一步移动时，首先检查当前解是否已经访问过。如果已经访问过，说明进入了某个循环，禁忌长度变为原来的 N_{IN} 倍；如果经过给定的若干次迭代后，没有重复的解出现，禁忌长度变为原来的 N_{DE} 倍。

为了避免循环，主动禁忌搜索算法给出了逃逸机制。搜索过程中，当大量解重复出现次数超过给定次数 R_{EP} 时，逃逸机制便被激活。逃逸操作一般通过从当前解执行若干步随机移动实现，执行移动的步长在定义域内随机选择。为了避免很快跳回刚搜索过的区域，所有随机操作都被禁止。

禁忌搜索算法使用历史记忆寻优，用禁忌表指导优化搜索，结合渴望水平，系统地实现了集中强化搜索和分散多样化搜索的平衡。而主动禁忌搜索算法则使用反馈策略和逃逸机制来加强这种平衡。因此，理论上说，主动禁忌搜索算法比一般的禁忌搜索算法效果更好，搜索的质量更高。

3. 主动禁忌搜索算法的基本步骤

主动禁忌搜索算法的核心思想是反馈策略与逃逸机制，上面只是给出了基本思想。实际应用中，反馈策略与逃逸机制有多种实现方法。例如：如果 num_dec 代内没有重复解出现，则禁忌表长度变为原来的 N_{DE} 倍；如果重复解出现的总次数（不是哪一个解重复的次数，而是所有解重复次数的和）达到 num_esc，则执行逃逸操作。主动禁忌搜索算法的基本步骤如下：

第 1 步：初始化两个计数器：$n_dec=0$，$n_esc=0$。

第 2 步：初始化其他参数，给定初始解。

第 3 步：针对当前解，给出候选解集。

第 4 步：根据禁忌表情况和渴望水平情况，选出一个解作为下一次迭代的初始解，更新记录表（包括正常的禁忌表和所有访问过的解）。

第 5 步：如果该选中的解出现过，则禁忌长度 $t=tN_{IN}$，$n_esc=n_esc+1$，$n_dec=0$；否则 $n_dec=n_dec+1$。

第 6 步：如果 $n_dec=num_dec$，则 $t=tN_{DE}$，$n_dec=0$。

第7步：如果 $n_esc = num_esc$，则实施逃逸操作，$n_esc = 0$，$n_dec = 0$。

第8步：如果满足停止准则，则算法终止；否则转第3步。

以上步骤主要用来说明主动禁忌搜索算法中提出的反馈机制和逃逸操作，至于常规禁忌搜索算法中包括的渴望水平与选择策略等，这里没有详细描述。主动禁忌搜索算法的流程图如图4.9所示。从中可以清楚地看到主动禁忌搜索算法的核心思想，看到逃逸机制的触发条件以及加大或者缩小禁忌长度的具体办法。

4. 主动禁忌搜索算法的内存管理

基本的禁忌搜索算法中，需要存储的内容比较少，禁忌表的长度一般不会很长。即便是这样的情况下，存储的空间与效率也比较重要，例如上文介绍的频数表和短期表共用一个矩阵的方法。实际上，关于节省存储空间，提高访问速度方面还有很多研究，本书没有详细介绍。

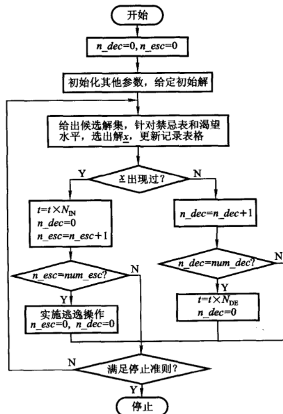


图4.9 主动禁忌搜索算法的流程图

主动禁忌搜索算法中，所有访问过的解都需要存储起来，这避免了搜索的循环，同时可以自动地调整禁忌长度，能很好地平衡集中强化搜索与分散多样化搜索。但是，事物都是具有两面性的，主动禁忌搜索算法中存储了大量的信息，而且对这些信息要频繁地访问，如果只是使用一般的数组来存储，显然是不够的。为了提高搜索的速度，主动禁忌搜索算法中一般使用哈希（Hashing）表来存储这些信息。

哈希技术即对关键字进行数学转换，得到一个位置信息，使得在数组或者文件的这个位置可以检索到这个数据的技术。直观理解，哈希技术就是对关键信息“切碎”而后进行管理的一种技术。数学上，哈希转换可以描述为映射$f:A\rightarrow B$，其中$|A|>|B|$，通过对较少的数据B检索完成对较大的数据A的检索，因此提高了检索的效率。同时，由于$|A|>|B|$，哈希函数f不是一一映射，而是多对一映射，所以有时会引起冲突，哈希技术中有专门关于避免冲突等方面的研究。关于哈希技术的更详细描述超出了本书的范围，有兴趣的读者可以参考相应关于数据结构的文献。

下面给出一个简单的例子来说明哈希函数。很多电台节目中要记录参与者的电话或者手机号码，而这些号码通常为10多位，当数据量很大时，检索起来比较困难；通常选取4位尾号作为关键字，使用哈希表进行管理，例如手机号码130××××1589映射为1589。如果应用得好，哈希技术可以大大提高数据访问的效率。

主动禁忌搜索算法中，存储的是解的表达（编码）方式，即映射中的A，而如果用解的主要一些配置信息作为关键字构成哈希函数，即映射中的B。通过检索这些配置信息，可以很快地找到相应的解。

除了使用哈希表来管理这些存储的解之外，还可以使用二叉树等方式，这里不赘述。

5. 主动禁忌搜索算法的应用

Battiti和Tecchiolli提出主动禁忌搜索算法的时候，针对**0-1**背包问题和大规模的二次指派问题（QAP）对算法进行测试，取得了很好的效果。而后，主动禁忌搜索算法已经在很多领域得到了成功的应用，包括旅行商问题（TSP）、车间调度、车辆路径问题（VRP）、神经网络参数调整以及电力系统中有功/无功控制等领域，也有人用来求解函数优化问题，国内关于主动禁忌搜索算法的研究与应用还比较少。

主动禁忌搜索算法主要思想是利用反馈机制自动调整禁忌长度和逃逸机制来避免循环。事实上，具有这样思想的禁忌搜索算法都可能称为主动禁忌搜索算法，而且主动禁忌搜索算法还可以包括其他方面关于禁

忌搜索算法的改进，例如并行的主动禁忌搜索算法等。

虽然主动禁忌搜索算法目前还处于发展阶段，但是可以预见，随着应用领域的扩大和研究的深入，主动禁忌搜索算法作为一种鲁棒性很强的元启发式算法，必将得到长足的发展。

4.5.3 禁忌搜索算法与遗传算法混合的搜索策略

近年来，混合优化策略得到了广泛的应用，并取得了很好的效果，其设计与分析已经成为算法研究的一个热点。本小节首先简要介绍混合优化策略的研究与应用情况，然后给出禁忌搜索算法和遗传算法的混合优化策略。

1. 混合优化策略

随着工程技术的发展和问题范围的拓宽，问题的规模和复杂度越来越大，传统算法的优化效果往往不够理想，同时算法理论研究的滞后也导致了单一算法性能改进程度的局限性。基于这种情况，算法混合的思想已经成为提高算法优化性能的重要且有效的途径。

近年来，有学者分析了遗传算法、禁忌搜索算法以及本书后面章节要介绍的模拟退火算法、蚁群优化算法、粒子群优化算法等元启发式算法的特点，并统称为广义的邻域搜索算法。广义邻域搜索算法是相对于梯度下降法等传统的邻域搜索算法而言的，为构造新的优化算法提供了一个框架，其中包括如下6个方面的要素：

①搜索机制是构造算法框架和实现优化的关键，是决定算法搜索行为的根本点。

②搜索方法决定着优化的结构，即每代有多少解参与优化。

③邻域函数决定了邻域结构和邻域解的生产方式。

④状态更新方式即如何从旧状态中确定新的当前状态，是决定算法整体优化特性的关键步骤之一。

⑤控制参数必须以一定的方式进行修改，以适应算法性能的变化。

⑥停止准则决定了算法的最终优化性能。

通过分析广义邻域搜索的关键要素，又提出了广义邻域搜索的统一结构，这对算法混合策略的研究以及设计新的算法具有一定的指导意义。

当前关于混合优化算法的应用已经比较广泛，其中混合进去的算法包括传统的优化算法以及各种启发式算法和元启发式算法。例如，模拟退火-单纯形算法混合优化策略、遗传-模拟退火算法混合优化策略、禁忌搜索-遗传算法混合优化策略等，还包括三种或者三种以上算法的

混合。应用领域包括函数优化、组合优化、神经网络设计等各个领域。

2. 禁忌搜索算法和遗传算法的局限性

禁忌搜索算法的优越性使这个算法得到了广泛的应用，同时禁忌搜索算法的一些局限性也促进了禁忌搜索算法与其他算法混合优化策略的产生与发展。众多禁忌搜索算法与其他算法的混合策略中，这里只介绍禁忌搜索－遗传算法混合策略。介绍之前，首先回顾一下这两种算法的缺陷。

（1）遗传算法的局限性

尽管遗传算法能够胜任任意函数高维空间组合优化问题，但是对于大规模神经网络的结构和权值的优化等超大规模的优化问题，遗传算法的应用就受到了限制。究其原因，主要是遗传算法每一代都要维持一个较大规模的种群，对整个种群的存储与访问占用了大量的空间和时间，当问题规模相当大时，这样大的时间和空间开销是无法接受的。

遗传算法还有一个缺点是“早熟”。造成遗传算法早熟的原因有两个：其一便是遗传算法中的交叉算子。交叉算子使得种群中的染色体之间具有局部相似性，可能导致搜索停滞不前。其二是遗传算法中的变异概率一般比较低，变异操作带来的种群多样性一般不够，而主要呈现出的是交叉操作带来的种群相似性。

此外，由于变异操作的“力度不够”，遗传算法的爬山能力一般较差，因此如何提高爬山能力也成为遗传算法的一个主要研究方面。

（2）禁忌搜索算法的缺憾

相对于遗传算法，禁忌搜索算法具有较快的收敛速度，但是禁忌搜索算法的搜索性能较大地依赖于给定的初始解。一个较好的初始解往往使禁忌搜索算法很快收敛于全局最优解，而一个较差的初始解可能极大的降低算法的收敛速度。因此，应用中往往使用其他启发式算法给出一个较好的初始解，来提高禁忌搜索算法的性能。

禁忌搜索算法的另外一个缺憾是算法的串行性。算法初始时是一个解而不是一个种群，迭代过程也只是从一个解移动到另外一个解，而不是一个种群到另外一个种群。相对于遗传算法等并行算法而言，禁忌搜索算法的串行性导致全局搜索能力有待提高，也正因为如此才出现了4.4.1节中并行禁忌搜索算法的出现。

3. 禁忌/遗传混合策略的基本思想

最早把记忆功能引入到遗传算法中的是Muhlenbein，从一个很宽广的范围对遗传算法和禁忌搜索算法进行了分析和比较，指出了二者进行混合的可能性以及理论基础，但并未提出具体的混合方法。Reeves把

禁忌搜索算法的多样化思想引入到遗传算法的交叉和变异中，并使得搜索过程具有记忆性，收到了很好的效果。而后，禁忌搜索-遗传算法混合搜索策略（简称禁忌/遗传混合策略）得到了较为广泛的应用。

禁忌/遗传混合策略中，由于遗传算法的广域搜索能力较强，一般作为“主算法”；由于禁忌搜索算法的局部搜索能力较强，一般作为“从算法”。主算法和从算法的概念并不是针对两个算法的重要性，而是这样的混合算法从整体看来比较像一般的遗传算法，而其中的实现方法上又带有禁忌搜索算法的思想。目前应用较多的禁忌/遗传混合策略主要包括两种形式，嵌入禁忌搜索的遗传算法和引入禁忌搜索思想的遗传算法，下面分别给出介绍。

（1）嵌入禁忌搜索的遗传算法

这种混合策略中，完整的禁忌搜索算法被嵌入在遗传算法中。遗传算法的每一步迭代中，要进行一次或者数次的完整禁忌搜索（而不是一次或数次禁忌搜索迭代）。遗传算法爬山能力比较弱主要是由于变异操作造成的，所以这种策略中一般使用禁忌搜索算法代替变异操作。使用禁忌搜索取代了标准变异算子之后，这个算子一般称为禁忌搜索变异算子，记为TSM算子。

这种使用了TSM算子的遗传算法的一般步骤可以描述为：

第1步：初始化算法，给出算法中各参数，并初始化种群。

第2步：判断遗传算法的停止准则是否满足。如果满足，停止算法，输出结果，否则继续以下步骤。

第3步：基于当前种群进行选择操作，例如使用轮盘赌方法。

第4步：进行交叉操作，更新种群和最优状态。

第5步：使用TSM算子变异：

子步骤5-1：对于每一个染色体，生成0，1之间的随机数r，如果$r \leq p_m$（其中p_m为变异概率），则对该染色体进行TSM变异，否则考虑下一个染色体。

子步骤5-2：初始化禁忌搜索算法，当前染色体即为初始解。

子步骤5-3：判断禁忌搜索算法迭代准则是否满足。如果满足结束禁忌搜索，进入第6步，否则继续以下步骤。

子步骤5-4：产生候选解集。

子步骤5-5：根据设定的渴望水平和禁忌表情况，选择一个解，并更新禁忌表。

子步骤5-6：转子步骤5-3。

第6步：以新的种群返回第2步，继续遗传算法。

嵌入禁忌搜索的遗传算法和一般遗传算法大部分都是一样的，所以上述步骤中没有对其中的编码方式、初始化、选择方法、交叉方法等环节做详细描述，这些环节的确定以及种群大小、迭代次数、交叉和变异概率等参数的给出已经在第3章详细介绍过了。

嵌入禁忌搜索的遗传算法中的禁忌搜索部分和一般的禁忌搜索也是一样的，包括其中的候选解集的确定、禁忌长度的给出以及渴望水平的选择等都没有特殊要求。但是，在这里禁忌搜索只是一个被嵌入的算法，主要用于邻域搜索。因此，算法的参数设置上可以适当加重邻域搜索的力度，例如一般不用中期表和长期表，迭代次数也不宜过大。

(2) 引入禁忌搜索思想的遗传算法

与嵌入禁忌搜索的遗传算法不同，这种混合策略只是把禁忌搜索算法的“禁忌”与“特赦”思想引入到遗传算法中来，对遗传算法中的交叉或者变异操作进行一定的改进。遗传算法的选择策略中有“精英保留”策略，主要是为了把性能良好的染色体直接保留到下一代。引入禁忌搜索思想后，不但可以实现“精英保留”，而且具有记忆功能，限制了个体被替换的频率。这种思想一般用于改进交叉算子，改进后的交叉算子称为TSR算子。

具有TSR算子的遗传算法可以描述为：

第1步：初始化算法，给出算法中的各参数，并初始化种群。

第2步；判断遗传算法的停止准则是否满足。如果满足，停止算法，输出结果，否则继续以下步骤。

第3步：基于当前种群进行选择操作，例如使用轮盘赌方法。

第4步：使用TSR算子进行交叉：

子步骤4-1：对于每一个染色体，生成0，1之间的随机数r，如果$r \leq p_c$（其中p_c为交叉概率），则该染色体被选中，否则没有选中。如此选出父代染色体。

子步骤4-2：对每对父代染色体进行交叉操作，产生两个子代。

子步骤4-3：以父代染色体平均适配值为渴望水平，以染色体的适配值为禁忌对象，禁忌表具有一定的长度。

子步骤4-4：如果子代染色体适配值优于渴望水平，则破禁，无论其是否被禁忌，该子代染色体都进入下一代；否则进入下一步。

子步骤4-5：如果子代染色体没有被禁忌，则该染色体进入下一代；否则进入下一步。

子步骤4-6：选择最好的父代染色体进入到下一代中。

第5步：进行变异操作。

第6步：以新的种群返回第2步，继续遗传算法。

可见，以上混合策略中，交叉操作采取了“禁忌”与“破禁”的思想，这是禁忌搜索的基本思想；但是又没有嵌入完整的禁忌搜索算法，没有使用诸如邻域移动等操作，所以称为“引入禁忌搜索思想的遗传算法”。

TSR算子的核心思想就是：交叉产生的子代中，对于比较优秀的，直接进入下一代，无论其禁忌状态如何，即达到了渴望水平（父代适配值的平均值）。对于一般的染色体，即不能达到渴望水平的，那么如果被禁忌了，宁愿选择父代染色体。这样可以保持种群的多样性，避免算法的早熟。至于经典遗传算法中讲的各种交叉算子，根据问题的需要可以随意选择，例如可以选择单点交叉或者双点交叉、与运算交叉或者或运算交叉等，这样的任何交叉算子引入了禁忌搜索思想后，都可以称作TSR算子。

禁忌/遗传混合策略还可以有其他的形式，例如同时使用TSR算子和TSM算子等。

4. 禁忌/遗传混合策略的应用情况

早在1993年，Fox等就混合禁忌搜索和遗传算法来模拟马尔可夫链，Glover等（1994）认为：尽管禁忌搜索算法和遗传算法有很大的区别，但有一些内在联系，并且可以在很大范围内混合两种算法来进行优化。而后，禁忌/遗传混合优化策略得到了广泛的应用，包括预警卫星传感器调度、防空作战中的目标分配问题、通信系统中的最佳多用户检测问题、带有时间窗的车辆路径问题、可变加工时间的工件调度问题、电力系统的资源分配和电压控制以及聚类分析等领域。而且，混合算法日益成为研究的热点，应用范围日益广泛。

4.5.4 其他改进方法

以上介绍了并行禁忌搜索算法、主动禁忌搜索算法以及禁忌/遗传混合优化策略，这些策略很好地改进了禁忌搜索算法的性能，应用的范围也比较广泛。除此之外，还有许多基于禁忌搜索算法的改进算法，这些算法也能较好地改进搜索性能，但大部分问题依赖性都比较强，应用范围因此受到一定限制，下面给出几个例子。

1. 基于其他方法构造初始可行解

上文已经多次提到，禁忌搜索算法的性能在较大程度上依赖于初始解的质量。为了提高算法收敛的速度，提高解的质量，很多场合下初始解不是随机生成的，而是基于其他算法给出的。

例如方永慧等人使用插入法生成高质量的初始解，然后利用禁忌搜索算法寻优；当禁忌搜索算法的最优解经过很多次迭代都不能得到改善时，基于当前解利用插入法重新构造搜索起点，从而能很快地跳出原来的搜索路径而从不同的方向进行搜索。经过典型优化问题 TSP 问题验证，这种基于插入法的混合算法具有很好的收敛性和寻优能力。贾永基等人也使用插入法构造初始解，进而使用禁忌搜索算法求解货运车辆调度问题。

这里提到的插入法（Insertion Method，简称 IM）是一种构造性启发式算法，最早是由 Rosenkrantz 等人为构造某一度量空间中的一条访问回路而提出，后来已经从算法复杂性的角度证明这种算法用来生成高质量的初始解时具有很大的优越性。

2. 快速局部搜索结合禁忌搜索的算法

当邻域规模比较大时，如果遍历整个邻域选择最优解然后根据禁忌表和渴望水平来完成一次迭代，则邻域搜索时间将比较长。这个情况下，可以将邻域的一部分作为候选解集来加快邻域搜索，也可以引入其他启发式算法来加快邻域搜索。

一种快速局部搜索（Fast Local Search，简称 FLS）算法将邻域空间划分为多个子邻域，并在邻域上设置活动标志。活动标志为 0 的子邻域称为活动子邻域，是待搜索的子邻域；活动标志为 -1 的子邻域称为不活动子邻域，是不用搜索的子邻域。初始时所有子邻域的标志都是 0，如果搜索完某个子邻域没有找到任何更好的邻居，那么该子邻域的活动标志置为 -1，否则该子邻域的活动标志保持为 0。随着解的不断改进，活动的子邻域越来越少，搜索的速度越来越快。直至所有子邻域活动标志都为 -1，则找到了局部最优解，邻域搜索结束。

贾永基等人将这种快速局部搜索算法引入到禁忌搜索算法中来，并巧妙地根据带时间窗车辆装卸货问题的特点，在客户和路径的对应关系上设置活动标志，将表示活动标志信息的矩阵和禁忌表合二为一。通过测试，在保证解的质量没有变化的情况下，搜索时间大大减少，表明这种混合禁忌搜索算法的有效性。

3. 带回访功能的禁忌搜索算法

Eugeniusz 提出了一种带回访功能的禁忌搜索算法，主要思想是每当历史最优解得到改进时，保存与这个解有关的信息。当算法迭代了预先给定的一定代数而最优解没有改进时，回跳到已经保存的有改进解的位置，取出与这个解有关的信息，从该点开始对未搜索的邻域进行搜索。重复上述步骤，直至搜索完所有区域。

这一方法的缺点是：在预先给定的步数内找到的很多解是重复的，浪费了很多的时间。为了解决这个问题，童刚等人引入哈希技术保存访问过的解，改进带回访功能的禁忌搜索算法，并针对 job - shop 问题验证了算法的性能，关于使用哈希技术管理访问过的解的具体方法这里不再详述。

4. 结合启发式的禁忌搜索算法

衣杨等人针对并行多机成组工件极小化通过时间的调度问题，提出了禁忌搜索结合启发式（记为 TSHEU）的算法，并对比了禁忌搜索结合分枝定界（记为 TSB&B）算法，表明两种算法都是有效的，而 TSHEU 算法速度更快。

并行多机成组工件极小化通过时间问题属于一类工件调度问题，其中成组调度的基本思想是：不同工件按其相似性分为若干个组；加工中，一个工件接在同组工件之后不需要重新准备，而接在不同组工件之后必须重新准备。这个问题属于 NP 难题，一般启发式算法难以用到具有实际意义的大型问题中。

针对单机流水时间问题的最优性条件和合理调度的性质（关于单机调度的这些定义与性质；本文略），衣杨等人提出了效果很好的启发式算法，并将该算法与禁忌搜索算法结合求解上面的多机问题。搜索过程中根据给出的启发式算法可以很容易地判断出某个解是否为邻域最优解，而不需要像一般问题那样通过遍历邻域来判断最优性，极大地节省了搜索时间。

5. 其他改进算法与混合算法

关于禁忌搜索的改进算法还有很多，可以对标准禁忌搜索算法的各个基本环节进行改进，包括初始解的确定方法、各参数的设定与调整方案、邻域搜索策略等。例如刘江华等人将极大似然加速算子引入到禁忌搜索算法中，每迭代一定次数，进行一次极大似然加速操作，也就是基于某种概率进行局部搜索，很大程度上加快了禁忌搜索的收敛。

禁忌搜索算法与其他算法的混合算法也还有很多，例如禁忌搜索算法与粒子群算法（PSO）混合、禁忌搜索算法与贝叶斯优化算法混合等。限于篇幅，在此不能一一列举，而且与禁忌搜索算法混合的有些算法可能要在本书后面章节中给出介绍。

4.4.4 节中介绍的各种改进或混合算法，一般具有较强的问题依赖性，其中的优化策略或者参数设定常常是根据特定的问题给出的，不容易移植到其他问题中，或者移植了效果也会下降。或者由于其他原因，这些算法到目前还没有得到很广泛的应用。

4.4.1～4.4.3节介绍的算法相对来讲通用性比较好。这些算法主要针对禁忌搜索算法本身的一些缺点进行改进，包括：针对串行搜索引入并行机制，针对禁忌长度的设定缺乏理论依据引入反馈机制，针对不能避免较大循环引入逃逸机制，等等。这些方面和待求解的优化问题无关，因此这些改进算法得到了较为广泛的应用，而且应用范围还有扩大的趋势。

此外，对于特定的问题而言，通用性好的算法一般不会是性能最好的，因为通用的算法不可能考虑给定问题的特征。如果问题有特殊的要求，例如计算时间要求很短，或者必须求得全局最优解，那么应该针对特殊问题的特殊要求，修改那些比较通用的算法，或者重新设计新的求解算法。例如中国象棋人机博弈中通常就采用穷举搜索，只是穷举方法上使用了分枝定界等技术而不是蛮力搜索而已。如果问题没有特殊的要求，则可以直接使用那些比较通用的优化方法求解。

4.6 禁忌搜索算法的应用

禁忌搜索算法是一种有效的组合优化求解算法，Glover最早也是针对组合优化问题提出的这种算法。但是，近年来禁忌搜索算法的应用范围得到了拓宽，包括函数优化（实优化）和多目标优化问题的求解等，下面分别给出介绍。

4.6.1 应用于实优化问题

1992年Hu首先将禁忌搜索算法扩展到函数优化领域，之后用禁忌搜索算法求解函数优化问题得到了一定的关注，而且现在还在发展之中。

1. 主要技术问题

使用禁忌搜索算法求解函数优化问题，首先要解决的问题是邻域的表征。函数优化中，当前解 x 的邻域通常定义为以 x 为中心、r 为半径的球，记为 $B(x, r)$，从而所有满足条件 $\|x'-x\| \leqslant r$ 的点 x' 都是 x 的邻域解，其中 $\|\cdot\|$ 表示范数。同时，为了使得 η 个邻域解在整个邻域内分布得比较均匀，可以再定义以 x 为中心、分别以 $r_1, r_2, \cdots, r_{\eta-1}$ 为半径的 $\eta-1$ 个同心球，将邻域分割为 η 个子邻域，在每一个子邻域中产生一个点。这样，x 的 η 个邻域解为 $\{x^i \mid r_{i-1} \leqslant \|x^i - x\| \leqslant r_i, i = 1, \cdots, \eta\}$，其中 $r_0 = 0$，$r_\eta = r$。

以上使用分割球的方法得到邻域解的过程中，计算量比较大，而且

邻域解的个数 η 不好给出。更为简便的做法是采用超立方体代替球，即对当前解 $x=(x_1, \cdots, x_n)^T$（n 为解的维数）的分量 $x_i(1\leqslant i\leqslant n)$ 做如下变换：

$$x'_i = x_i + s \tag{4.8}$$

其中，s 为步长。根据需要，s 可以随迭代步数的变化而变化，也可以针对不同的分量取不同的值。

此外，由于状态（或状态变化）以及适配值（函数值）是连续的，关于是否被禁忌的判断与求解离散问题也不同。这里，通常在禁忌对象的一定范围内都认为是禁忌的，例如禁忌对象的 $\pm 0.01\%$ 等。

2. 简化禁忌搜索算法用于实优化

所谓简化禁忌搜索算法，即不考虑所有邻域解都被禁忌的情况和渴望水平。使用简化禁忌搜索算法求解实优化问题 $\min f(x)$，$x\in \boldsymbol{E}^R$ 的基本步骤可以描述为如下。

第1步：随机给出初始可行解 x，历史最优解 $x^*=x$，迭代次数 $k=0$。

第2步：如果达到最大迭代次数，输出结果，停止算法，否则继续以下步骤。

第3步：产生邻域 D，并计算邻域解的适配值。

第4步：选择不被禁忌的最好解 $\underline{x}$，$x=\underline{x}$，$k=k+1$，更新禁忌表。

第5步：如果 $f(\underline{x})<f(x^*)$，则 $x^*=\underline{x}$；否则继续以下步骤。

第6步：返回第2步继续搜索。

从以上步骤可以看出，简化禁忌搜索算法中以设置最大代数为停止准则，以选中的解为禁忌对象，当然，如果某个解与禁忌表中某元素的差小于给定范围，也认为是被禁忌的。这样的算法简单明了，容易实现，而且与用于组合优化的禁忌搜索算法也十分相似，主要的区别是其中邻域的确定。

这个算法中，按式（4.8）给出邻域，其中的步长 s 按

$$s^{k+1} = rs^k \tag{4.9}$$

规律变化，其中，k 为迭代次数，r 为给定常数，第 k 次迭代的 s^k 与模拟退火算法中的 S 的含义类似，$s^k=cs^{k-1}$，其中 $0<c<1$ 为步长的减小比例。

施文俊等人认为这个算法中的步长 s 下降速度过慢，导致搜索广度大而精度不足。于是将整个迭代过程分为数轮，每经过一轮 s 的值减少一个数量级，采用分轮速降的方法协调广度与精度的矛盾，最后使用化工邻域中的换热网络优化问题进行了验证。

3. 具有自适应机制的禁忌搜索算法用于实优化

集中性搜索与多样性搜索始终是禁忌搜索算法运行的焦点，这在上文已经多次提到。集中性搜索用于对当前搜索到的较好解的邻域进行进一步的搜索，以便达到全局最优解；多样性搜索用于拓宽搜索区域，尤其是没有搜索过的区域。当搜索陷入局部最优时，多样性搜索能改变搜索方向，跳出局部最优解。集中性搜索与多样性搜索是重要的，但又是矛盾的，如何协调这对矛盾是应用禁忌搜索算法的一个难点。针对这个问题，已经开展了大量的研究工作，例如前文介绍的主动禁忌搜索算法利用反馈机制调整禁忌长度来协调集中性与多样性。

贺一等人提出了另一种协调集中性与多样性的策略。这种自适应策略中，将邻域和候选解集分为两个部分，一部分是集中性元素，用于集中搜索；另一部分是多样性元素，用于多样性搜索。邻域中集中性元素的产生办法和一般禁忌搜索算法中邻域的产生办法相似，而多样性元素则不同，常常是随机产生。候选解集中的集中性元素按最优性选取，而多样性元素则仍随机选取。

算法开始之前，候选解集中的集中性元素占元素总数的一半。迭代过程中，集中性元素个数 DL 动态地改变。如果本次迭代得到的当前最优解优于上一次迭代得到的最优解，则 $DL = DL + 1$；如果本次迭代得到的最优解等于或劣于上一次迭代得到的最优解，则 $DL = DL - 1$。另外，无论什么时候，候选解集中都要存在集中性元素和多样性元素，至少保留一个。

当解的质量有提高时，候选解集中的集中性元素增多，相应的进行集中性搜索的概率也增大；反之，当解的质量没有提高时，候选解集中的多样性元素增多，相应的多样性搜索的概率也增大。这样，根据搜索中的解的具体情况动态地调整集中性搜索与多样性搜索的比例，较好地解决了这一对矛盾。针对 TSP 问题，将具有这种自适应能力的禁忌搜索算法与其他文献上的神经网络的算法进行了比较，多数情况下这种禁忌搜索算法优于神经网络算法。

由于这种策略不需要问题的特殊信息，很容易应用于其他问题的求解。贺一等人用带这种策略的禁忌搜索算法优化神经网络中的权值和阈值。其中一些关键环节介绍如下。

(1) 禁忌对象：优化问题的目标函数为神经网络实际输出与目标输出的相对平均偏差，即

$$f = \frac{1}{n}\sum_{i=1}^{n}\frac{|T(i) - O(i)|}{|T(i)|} \times 100\% \tag{4.10}$$

其中，n 为样本点个数，$T(i)$ 和 $O(i)$ 分别为第 i 个样本点的目标函数值和实际输出值。以目标函数值 f 为禁忌对象，当候选解的目标函数值在禁忌表中某元素周围的一个很小的区间（例如 ±0.01%）时即被禁忌。

（2）邻域结构：在神经网络当前权值和阈值（即当前解）每一个元素的基础上加上一个修正量，形成一个邻域解。最大修正量记为 max_offset，修正量在这个最大值之内均匀地随机生成。

（3）自适应策略：对于集中性元素，最大修正量取很小的值，例如 ±0.05 ~ ±0.1，便于在该区域集中搜索；对于多样性元素，最大修正量取较大值，例如取 ±0.4 ~ ±2.0。其他方面完全按上述自适应策略实现。

使用这种禁忌搜索算法训练神经网络，与用 BP 算法训练神经网络做了比较，测试函数为正弦函数和 sinc 函数 $f(x)=\sin(x)/x$，结果这种算法的优越性十分明显。

这种禁忌搜索算法的核心思想是引入了集中性元素和多样性元素的概念，并根据搜索的具体情况自动调整两种元素的比例，很好地解决了集中性与多样性之间的矛盾。

4. 增强连续禁忌搜索算法

Chelouah 等（2000）提出了一种增强连续禁忌搜索算法，记为 ECTS。和前文介绍的具有自适应机制的禁忌搜索算法相似，ECTS 也尤其强调集中搜索和分散化搜索的重要性。简要地讲，ECTS 框架包括 3 个主要部分，即分散搜索、最有希望区域搜索、集中强化搜索。因此，增强连续禁忌搜索算法的主要步骤可以描述为图 4.10 所示的流程图。算法的这 3 个主要步骤是连续的，前一个步骤的结果为后一个步骤做准备，或者可能就是后一步要搜索的范围（定义域），后一步搜索是前一步的继续和强化。算法的 3 个主要步骤又比较相似，每一个主要步骤都可以认为是一个独立的禁忌搜索算法。下面对这 3 个主要部分分别做介绍。

（1）分散化搜索

分散化搜索中除了禁忌表外，又引入了“希望表”的概念。禁忌表中存放过去一定次数迭代中接受的邻域解，而希望表中存放搜索到的有希望区域。首先给定初始解，按前面介绍的分割超方体的方法得到规定个数的邻域解，选择那些既不在禁忌表中又不在希望表中的最优邻域解，作为下一次迭代的当前解。如此反复搜索，当目标函数出现了不可接受的恶化时，便开始搜索新的有希望区域，而这时当前解

就被认为是这个有希望区域的中心，记录在希望表中。希望表和禁忌表相似，也有一个给定的长度，当新的有希望区域进入希望表时，最差的（注意：不是最早的）有希望区域从希望表中退出。如果规定步数内没有发现新的有希望区域，则停止分散化搜索，进入最有希望区域搜索。

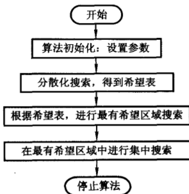


图 4.10 ECTS 的基本流程

注意：产生邻域时使用的超方体方法，便于实现；而判断一个解是否被禁忌时使用的是禁忌球方法，即判断该解与禁忌对象的距离是否在给定距离之内，这两者是有区别的。

可以看到，这个过程就是一个禁忌搜索，而且是比较复杂的禁忌搜索。其中希望表中存储的可以理解为算法的结果，只是这里不只保留一个结果，而是保留了规定个数（希望表长度）个最优结果。禁忌表和希望表的同时使用，有效地激励搜索远离初始点，避免迂回搜索，从而很好地实现了分散化搜索。

（2）最有希望区域搜索

分散化搜索中已经得到了一个希望表，这是“最有希望区域搜索”的开始区域，禁忌表已经没有意义。最有希望区域搜索的主要步骤如下。

第 1 步：计算希望表中所有解的目标函数值的平均值。

第 2 步：删除希望表中目标函数值高于平均值的解，即比较差的解。

第 3 步：将禁忌球半径和超方体邻域大小减半，对留下来的有希望解执行“产生邻域解，选出最优解”的操作过程。如果最优解优于产生它的那个有希望解，则使用该最优解替换当初的有希望解，否则不替换。如此对整个希望表扫描完毕。

第4步：如果还剩下多个有希望区域，转第1步继续搜索；否则结束。

（3）集中搜索

集中搜索是针对“最有希望区域搜索”得到的最有希望区域进行的，是又一轮的搜索。集中搜索的步骤可以描述如下：

第1步：清空禁忌表，初始化迭代步数等参数。

第2步：对当前解执行“产生邻域解，选出最优非禁忌解，更新禁忌表”的操作。

第3步：如果目标函数值在规定迭代次数内得到了改善，转第2步反复搜索；否则继续以下步骤。

第4步：如果目标函数值在规定迭代次数内没有改善或者达到了规定的迭代步数，则算法停止；否则将超方体邻域大小和禁忌球半径减半，转第1步反复搜索。

可以看到，这只是一个非常简要的描述，其中一句话“产生邻域解，选出最优非禁忌解，更新禁忌表”几乎相当于标准禁忌搜索算法的全过程。因此，分散化搜索，最有希望区域搜索、集中搜索都是很复杂的禁忌搜索过程，而整个的ECTS框架的复杂程度可想而知。

当然，ECTS框架中具有为数众多的参数，包括一些初始化参数和控制参数，而这些参数中有些需要用户给出，有些需要通过计算得到，有些参数需要根据问题规模设定，等等。关于这些参数的详细设置方法本书不展开讨论，有兴趣的读者可以查阅相关文献。

禁忌搜索算法用于实优化，可以是非常简单的基本禁忌搜索算法，也可以是很复杂的ECTS框架。对于特定的问题，读者需要根据具体情况灵活运用禁忌搜索算法来解决。

4.6.2　应用于多目标优化问题

现实生活中，很多问题都具有不止一个目标函数，称为多目标优化问题，本小节介绍禁忌搜索算法应用于多目标优化的情况。

1. 多目标优化问题

一般的，多目标优化问题可以描述为

$$\min/\max\ F(\boldsymbol{X}) \tag{4.11}$$

$$\text{s.t. } \boldsymbol{X}\in\boldsymbol{S}=\{\boldsymbol{X}\mid \boldsymbol{X}\in\boldsymbol{A}^n, g_i(\boldsymbol{X})\leqslant a_i, h_j(\boldsymbol{X})\leqslant b_j\}\ i=1,\cdots,m, j=1,\cdots,n$$

其中，决策变量 $\boldsymbol{X}$ 为 n 维向量，$F(\boldsymbol{X})=(f_1(\boldsymbol{X}),\ \cdots,\ f_k(\boldsymbol{X}))$ 为 k 维目标函数向量，$\boldsymbol{S}$ 为可行解的集合，带有 m 个不等式约束和 n 个等式约束，a_i和 b_j为常数。对于连续型变量，$\boldsymbol{A}=\mathbf{R}$；对于离散型变量，$\boldsymbol{A}$ 为

可行取值的集合。目标函数 $F(X)$ 可以是极大化，也可以是极小化，但是两者容易转化，为了讨论问题方便，下面讨论中只考虑极小化情况，即 $\min F(X)$，除非有特殊说明。

多目标优化问题中的目标通常是矛盾冲突的，这导致问题的最优解通常不止一个，而是多个，而这样的“最优解”也不是通常意义上的最优解，而是 Pareto 最优解（优化解）。Pareto 最优解，也称为非受控（Non - Dominated）解，定义如下：

一个解 $X^* \in S$ 是 Pareto 最优解，当且仅当不存在 $X \in S$ 满足

①$f_i(X) \leqslant f_i(X^*)$，$i=1, \cdots, k$；且

②$f_i(X) < f_i(X^*)$，$\exists i \in \{1, \cdots, k\}$

换句话说，如果没有一个解能改善目标函数的某个分量而不破坏任何一个分量，那么这个解就是 Pareto 最优解。既然没有哪个解能比 Pareto 最优解更优，求解多目标优化问题时就应该寻找尽可能多的 Pareto 最优解。

2. 多目标问题求解方法

传统的求解多目标问题的方法具有一定的局限性，这是与这些方法的本质与生俱来的。传统方法通常依赖于目标函数的类型（如线性或非线性等）以及决策变量的类型（如整数或者实数）等。影响算法性能的因素很多，包括解空间的规模、约束和决策变量的个数，以及解空间的结构（如凸的或非凸的）等。到目前为止，没有一种传统的方法能求解任何变量类型、任何约束和目标函数形式的多目标问题。例如，单纯形法只能求解目标函数和约束变量都是线性的情况，几何规划只能求解多项式形式或者符号变量目标函数的问题。然而，现实中的多目标问题常常具有多种变量形式、复杂的目标函数以及约束条件，传统的方法不能满足实际问题的需要。

在过去，实际问题必须描述成一种特定的传统方法能够求解的形式，然而这是不容易的。为了能够求解，很多实际问题需要经过大量的假设或者修改，例如：变量的取整（离散化），放松约束，某种程度上做近似处理，等等，这必然会影响到解的质量。为了解决传统优化方法中的这些问题，提出了一系列不依赖于问题的启发式算法，例如本书介绍的禁忌搜索算法、遗传算法、模拟退火算法等。这个时候，只需要对这些算法做少许修改，就可以解决各种实际问题，Adil 等把这个过程描述为图 4.11 所示的形式。

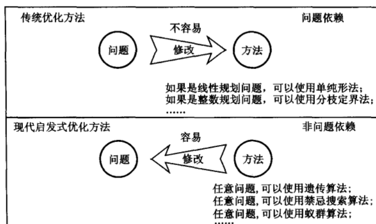


图 4.11　现代算法与传统算法的区别

使用禁忌搜索算法求解多目标问题，最简单的一个思路就是通过引入权重机制，将多目标问题化为单目标问题，然后求解。这种做法完全属于多目标问题的处理，求解所用的禁忌搜索算法与普通的禁忌搜索算法没有任何区别。Gandibleux 等（1997）提出了一种基于权重和分级函数（Scalarizing Functions）的多目标禁忌搜索算法，并用来求解组合优化问题。Hansen（2000）提出一种包含多个解向量的禁忌搜索算法，每个解向量都有自己的禁忌表，并引入权重来引导搜索达到非支配面。Ehrgott和 Gandibleux 关于应用禁忌搜索算法求解多目标问题给出了精彩的综述，这里不展开讨论，只在下面小节中介绍一种非常巧妙的多目标禁忌搜索 MOTS 算法。

3. 多目标禁忌搜索 MOTS 算法

禁忌搜索算法在每一步迭代过程中，都要产生多个邻域解，从中根据禁忌准则和渴望水平来产生下一步迭代的初始解，正是受这个特点的启发，Baykasoglu 等（1999）提出了一种新的多目标禁忌搜索 MOTS 算法。Baykasoglu 等认为：类似禁忌搜索算法这样，凡是在求解的过程中要同时处理多个解的算法，包括每代都保留一个种群的遗传算法，都可以很容易用来求解多目标优化问题。

MOTS 中，除了基本禁忌搜索算法中的禁忌表之外，引入了另外两个表，分别是 Pareto 表（Pareto List）和候选表（Candidate List）。Pareto 表用来收集搜索到的 Pareto 最优解，其中的 Pareto 最优解都曾经用来作为种子解（Seed Solution）来产生邻域解。候选表用来暂时存放搜索到的其他非受控解，如果在后续的迭代中，它们仍然保持“非受控”

状态，可能作为种子解而进入 Pareto 表。MOTS 算法与一般禁忌搜索算法的其他主要区别是解的选择与更新策略，下面对 MOTS 算法中几个关键环节给出介绍。

（1）初始解

和其他禁忌搜索算法一样，随机给出或用其他算法产生一个可行初始解。

（2）邻域解的产生

根据变量性质的不同，例如连续的、离散的、**0-1** 变量等，邻域解的产生方法不同，这与一般禁忌搜索算法没有什么区别。但是有一点不同，产生的这给定个数的邻域解必须是不受控于种子解（当前解）的，因为搜索的目标就是要寻找 Pareto 解，当然这些邻域解也要非禁忌的。要达到这一点，可以使用最简单的产生方法，也可以使用其他复杂的策略，如变结构策略等。

这种邻域产生原则相当于不接受劣解，而用下面介绍的候选表来避免循环。这一点与普通禁忌搜索算法是不同的，普通禁忌搜索算法的一个主要特点就是接受劣解。

（3）种子解的选择

选择种子解的核心策略是 Pareto 最优性策略。Pareto 最优性是一个经济学上的概念，直观地可以这样理解：如果一个解是 Pareto 最优的，那么没有如下这样的解存在：至少有一个分量比这个解更优，而没有一个分量比这个解更差。种子解的选择包括如下步骤。

第 1 步：对于每个邻域解，计算目标函数值。

第 2 步：从邻域解中选择候选解，候选解要求相对于所有其他邻域解、Pareto 表中的解和候选表中的解是 Pareto 最优的。

第 3 步：从候选解中随机选择一个作为种子解。如果没有候选解，则从候选表中选择一个最“老”的（最早进入候选表的）解作为种子解。

（4）各种表的更新

当搜索开始时，初始解作为 Pareto 最优解放入 Pareto 表中。迭代过程中，相对于其他邻域解而言，不再具有 Pareto 最优性的解从 Pareto 表（或候选表）中移除。选中的种子解加入到 Pareto 表中，而如果还有其他候选解，则加入到候选表中。以种子解为禁忌对象，关于禁忌表的设置和操作与普通禁忌搜索一样。

Pareto 表和候选表的长度没有限制，这一点与禁忌表是不同的。任何一个解不会因为在这两个表中时间太长了而退出，凡是被移除的解，

都是因为已经不再是 Pareto 最优解了，Pareto 表和候选表以及邻域中都是这样。

(5) 渴望水平

一般的禁忌搜索用于离散问题的时候，禁忌对象不是解本身，而是解的某些属性，或者称为状态改变。这种情况下，同样的禁忌对象，作用于不同的当前解，得到的是不同的新解，完全可能带来优于历史最优解的解。为了避免漏掉这样的解（或者叫做搜索区域），需要设置渴望水平。而如果将整个解作为禁忌对象，同样的解不可能带来不同的目标函数值，没有必要设置渴望水平，禁忌搜索算法用于实优化时通常都是这样。

(6) 停止准则

如果达到了预先设定的最大迭代次数，或者候选表为空算法无法找到下一个种子解，则算法停止。

为了进一步说明种子解的产生和各种表的更新，给出一个简单的例子。两个目标函数需要最大化，决策变量为二维实数，邻域大小取 3。设当前种子解向量为（4.8，4.6），目标函数值向量为（52.40，40.93）；产生的 3 个邻域解为（6.3，6.1）、（6.0，6.0）和（6.4，6.0），对应目标函数值分别为（60.08，47.09）、（58.79，46.54）和（60.39，46.86），可见所有邻域解相对于种子解都是 Pareto 最优的。下面选择候选解。从 3 个邻域解来看，目标函数值为（58.79，46.54）的解不是 Pareto 最优的（两个分量都劣于其他解），划掉；另外两个不相互受控。如果再对照 Pareto 表和候选表，目标值为（60.08，47.09）和（60.39，46.86）的邻域解还是 Pareto 最优的，则这两个都是候选解。将 Pareto 表和候选表中所有不再具有 Pareto 最优性的解移除，从这两个候选解中任意选择一个作为种子解加入 Pareto 表中，另一个加入候选表中。此外，选出的种子解还要加入禁忌表中，禁忌表中最“老”的元素退出禁忌表，这和一般禁忌搜索算法一样。

MOTS 算法的计算流程如图 4.12 所示。

与其他应用禁忌搜索算法求解多目标优化问题的方法不同，这个 MOTS 算法框架不需要任何关于各目标函数之间的权重信息，不需要将多目标函数化为单一目标函数，而是直接利用禁忌搜索算法的结构进行搜索。而且，其他应用于多目标的禁忌搜索算法都比普通禁忌搜索算法（单目标）额外需要一些参数，而 MOTS 算法没有任何额外的参数需要设置。

如果 MOTS 算法用于实优化，则生成邻域的步长和邻域的大小对算

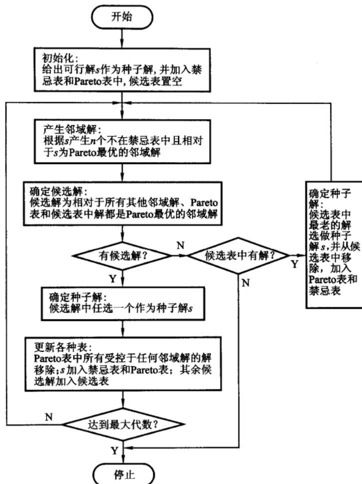


图 4.12　多目标禁忌搜索算法的流程图

法性能很重要。如果变量范围很宽而步长很小，则搜索时间会明显增加。相反，如果步长很小，则容易跳过最优解。实际上，这对于单目标的实优化也是相似的。

有一点值得关注：当算法收敛时，会出现邻域中找不到具有 Pareto 最优性的候选解且候选表也为空的情况而自动停止。因此，可以放心地设置足够大的迭代次数来保证算法收敛。

Baykasoglu 等提出 MOTS 算法时是应用于离散多目标的，而后又应用于连续优化，并经函数测试，取得了比用遗传算法求解多目标问题更好的效果。这种 MOTS 算法在国外已经有一些应用，但国内应用得还比

较少。但是，作为一种新颖高效的解决多目标优化问题的方法，这个框架很可能会得到很好的发展。

4.6.3　电子超市网站链接设计中的应用

禁忌搜索算法作为一种不依赖于问题的高效寻优算法，在工程实践中已经得到广泛的应用。下面给出一两个实际应用的举例，以加深读者对禁忌搜索算法的理解。

1. 电子超市网站链接结构的优化问题

电子超市是 BtoC 电子商务的一种表现形式，经营电子超市的公司主要通过网站进行产品的宣传和交易。为满足不断变化的需求，网站需要跟踪顾客的行为，并适时调整网站的链接结构，才能在竞争中保持有利地位。关于网站结构的设计已经有一些学者进行了研究，下面给出一个更新和优化网站结构的模型，并使用禁忌搜索算法进行求解。

电子超市网站中的链接主要分为两类，反映商品目录结构的链接称为基本链接，方便顾客浏览的链接称为附加链接。如果将网页和链接分别视为顶点和弧，则网站结构可以抽象为一个带标号的有向图。设网站中共有 N 个网页，标号为 $0 \sim N-1$，其中主页的标号为 0。定义布尔矩阵 $\boldsymbol{B}=\{b_{ij} \mid i, j=0, \cdots, N-1\}$，其中 $b_{ij}=\mathbf{1}$ 表示链接 (i, j) 为基本链接，$b_{ij}=\mathbf{0}$ 表示其他情况。定义布尔矩阵 $\boldsymbol{X}=\{x_{ij} \mid i, j=0, \cdots, N-1\}$ 代表网站的一种链接结构，其中 $x_{ij}=\mathbf{1}$ 代表链接 (i, j) 存在，$x_{ij}=\mathbf{0}$ 代表链接 (i, j) 不存在。

链接不存在长短的差异，所以网站结构图为一个无权图。网页的层次定义为从主页到达网页经过的最少链接个数。

2. 相关的基本概念

(1) 链接的可达性

链接的可达性取决于其所在网页的情况，定义为顾客点击链接的可能性。在不对链接进行特殊处理且不考虑顾客偏好的情况下，某网页上各个链接被点击的可能性是相同的，这样链接 (i, j) 的可达性

$$H_{ij}(\boldsymbol{X}) = \frac{x_{ij}}{\sum_{k=0}^{N-1} x_{ik}}, i,j = 0,\cdots,N-1 \tag{4.12}$$

(2) 网页可达性

网页的可达性定义为用户沿所有路径到达此网页的可能性之和。由于实际网站中网页和链接的数目众多，计算所有可能的路径几乎是不可能的，这里考虑主要的路径，即用户按照网页层次由浅入深的路径，这

样的路径一定包含了由主页到达页面的最短路径。为得到这样定义的到达每个网页的所有路径，可以使用路径树生成算法生成路径树。

根据得到的路径树，可以计算下列物理量：

$$N_i = f_i(\boldsymbol{X}), i = 1, \cdots, N-1 \tag{4.13}$$

$$L_{il} = g_{il}(\boldsymbol{X}), i = 1, \cdots, N-1; l = 1, \cdots, N_i \tag{4.14}$$

$$J_{ilj} = h_{ilj}(\boldsymbol{X}), i = 1, \cdots, N-1; l = 1, \cdots, N_i; j = 1, \cdots, (L_{il}+1) \tag{4.15}$$

其中，N_i 为用户可以到达网页 i 的路径条数；L_{ij}为到达网页 i 的第 l 条路径所需的步数；J_{ilj}为到达网页 i 的第 l 条路径的第 j 个网页的标号。于是网页 i 的可达性计算公式为

$$P_i(\boldsymbol{X}) = \sum_{l=1}^{N_i} \prod_{j=1}^{L_{il}} H_{J_{ilj}J_{ilj+1}}(\boldsymbol{X}), i = 1, \cdots, N-1 \tag{4.16}$$

假设顾客都是首先到达网站主页，因此不定义主页的可达性。

（3）平均载入时间

设 $\tau_i(i=0,\ \cdots,\ N-1)$ 为网页 i 的平均下载时间，它与网页大小和网络的平均传输速度有关。网页 i 的平均载入时间 T_i与到达此网页的路径条数和路径上所有网页的载入时间有关，计算公式为

$$T_i(\boldsymbol{X}) = \frac{1}{N_i}\sum_{l=1}^{N_i}\sum_{j=1}^{L_{il}+1} \tau_{J_{ilj}}, i = 1, \cdots, N-1 \tag{4.17}$$

（4）网页访问率

根据网络服务器在过去一段时间内统计的每一网页被访问的次数，可以按式（4.18）计算网页访问率

$$Q_i = \frac{V_i}{\sum_{i=1}^{N-1} V_i}, i = 1, \cdots, N-1 \tag{4.18}$$

其中，Q_i为网页 i 的访问率；V_i为过去一段时间内网页 i 被访问的次数。

3. 数学模型

（1）模型的建立

按照方便顾客的原则，访问率高的网页应该具有较大的可达性和较小的载入时间，这里采用相关性来度量两组量 U_i，$V_i(i=1,\ \cdots,\ N-1)$ 的相关程度：

$$\underset{i=1}{\overset{N-1}{Cov}}(U_i, V_i) = \frac{\sum_{i=1}^{N-1}(U_i - \overline{U})(V_i - \overline{V})}{\sqrt{\sum_{i=1}^{N-1}(U_i - \overline{U})^2 \cdot \sum_{i=1}^{N-1}(V_i - \overline{V})^2}} \tag{4.19}$$

为保持网站整体结构的稳定性，在每个页面上增加或减少的链接个数不

能太多，同时，应该避免新增加的链接过多地集中于某些网页，或者减少链接时将指向某些网页的链接过多地删除。另外，新增加的链接在内容上应该具有相关性。于是建立如下多目标数学模型：

$$\max f_1(\boldsymbol{X}) = \mathop{Cov}_{i=1}^{N-1}(Q_i, P_i(\boldsymbol{X})) \tag{4.20}$$

$$\max f_2(\boldsymbol{X}) = \mathop{Cov}_{i=1}^{N-1}(Q_i, (T_i(\boldsymbol{X}))^{-1}) \tag{4.21}$$

$$\text{s. t.} \sum_{j=0}^{N-1} |x_{ij} - a_{ij}| \leqslant R_i, i = 0, \cdots, N-1 \tag{4.22}$$

$$\sum_{j=0}^{N-1} |x_{ji} - a_{ji}| \leqslant C_i, i = 0, \cdots, N-1 \tag{4.23}$$

$$x_{ij} - b_{ij} \geqslant 0, i, j = 0, \cdots, N-1 \tag{4.24}$$

$$u_{ij} - x_{ij} \geqslant 0, i, j = 0, \cdots, N-1 \tag{4.25}$$

其中，式（4.20）表示最大化网页可达性与网页访问率的相关性，网页访问率 Q_i 为常数，调整后网站链接结构 $\boldsymbol{X}$ 为决策变量；式（4.21）表示最大化网页载入时间的倒数与网页访问率的相关性；式（4.22）表示增加或减少指向某个网页的链接的个数约束，常数 a_{ij} 为网站的当前链接结构，R_i 为常数；式（4.23）表示某网页上增加或减少的链接个数约束，C_i 为常数；式（4.24）表示基本链接不可以删除，常数 b_{ij} 表示基本链接情况；式（4.25）表示增加的链接内容上应该具有相关性，常数 $u_{ij}=1$ 表示网页 i 和 j 内容上相关，$u_{ij}=0$ 表示网页 i 和 j 内容上不相关。

（2）转化为单目标问题

设多目标问题式(4.20)～(4.25)的理想点为（f_1^*，f_2^*），并设 $E(\boldsymbol{X})$ 为式(4.22)～(4.25)构成的可行域。采用处理多目标问题的极大模理想点法，上述问题可以转化为单目标问题：

$$\min \lambda \tag{4.26}$$

$$\text{s. t. } X \in E(\boldsymbol{X}) \tag{4.27}$$

$$f_1^* - f_1(\boldsymbol{X}) \leqslant \lambda \tag{4.28}$$

$$f_2^* - f_2(\boldsymbol{X}) \leqslant \lambda \tag{4.29}$$

$$\lambda \geqslant 0 \tag{4.30}$$

根据极大模法相关定理，问题式(4.26)～(4.30)的最优解为原问题式(4.20)～(4.25)的弱有效解。于是，求解原多目标问题变为求解其理想点和新的单目标问题。

4. 求解算法

网站链接结构优化问题的模型中，要用到路径树生成算法，使得

网页的可达性和载入时间无法用解析形式来表达，故难以用传统的优化方法求解。这里使用禁忌搜索算法来求解原多目标问题的理想点以及转化后的单目标问题。求解过程中的相关环节比较相似，集中说明如下。

（1）编码方式：决策变量为 0－1 变量，所以直接采用 0－1 编码方式。

（2）初始解：算法的初始解取网站的当前链接结构，即 $\boldsymbol{X}=\boldsymbol{A}=\{a_{ij}\}$。容易知道，这样的初始解是可行的。

（3）邻域结构：任意改变当前解 $\boldsymbol{X}$ 中的一个元素的值形成的解构成当前解的邻域，只保留满足约束条件的邻域解，即可行邻域解。

（4）禁忌表：以发生改变的元素 (i,j)，即增加或删除的链接为禁忌对象，禁忌长度取常数 7。此外，引入中期表来记录修改（增加或删除）链接 (i,j) 的频数，并施加频数惩罚，$f'(\boldsymbol{X})=f(\boldsymbol{X})-w\cdot penalty(i,j)$。

（5）渴望水平：当优于历史最优解时，就认为达到了渴望水平。

（6）停止准则：设置最大迭代次数为停止准则。

当然，完全可以使用本章介绍的多目标禁忌搜索 MOTS 算法直接求解这个多目标问题。只是为了说明简单的禁忌搜索算法就能很好地求解如此复杂的优化问题。传统的优化方法只能求解特定类型的问题，例如线性的、二次的等；而禁忌搜索算法能求解的问题没有这样的限制，其中变量甚至可以不是解析的，只要能用程序得到变量的取值就可以了，充分显示出禁忌搜索算法极大的灵活性。

5. 计算举例

图 4.13 所示为一个网站的基本链接结构，附加链接见表 4.1，在过去一段时间内每个网页的访问次数见表 4.2，平均加载时间见表 4.3。表 4.4 中给出了矩阵 $\boldsymbol{U}$ 中为 0 的元素集合，即不相关网页对。

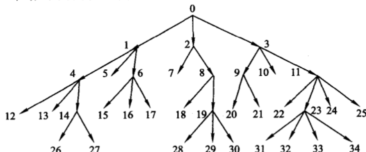


图 4.13 网站的基本链接结构

用Java语言实现了以上算法，在每个网页上最多增、减两个链接（$R_i=C_i=2$，$i=0$，1，…，$N-1$）的情况下进行了仿真实验，得到的仿真结果见表4.5。结构调整后目标函数值f_1由0.381增加到0.736，f_2由0.251增加到0.672，改进效果明显。

表4.1 优化前网站的附加链接

(0,4)	(0,18)	(0,23)	(0,26)	(2,6)	(2,17)	(8,16)	(9,14)	(15,27)	(17,30)
(20,31)	(21,34)	(23,30)	(25,26)	(26,0)	(26,3)	(27,1)	(27,3)	(28,1)	(28,4)
(29,3)	(30,0)	(30,5)	(31,5)	(31,7)	(32,5)	(32,7)	(32,8)	(33,9)	(34,6)

表4.2 网页的访问次数

v_i	0	1	2	3	4	5	6	7	8	9
—	120	19	38	36	72	52	61	33	15	23
10 +	23	13	55	63	34	43	15	16	7	12
20 +	21	38	14	64	24	35	92	51	11	22
30 +	62	41	32	22	36					

表4.3 网页的平均加载时间

τ_i/s	0	1	2	3	4	5	6	7	8	9
—	5.28	3.64	0.26	6.20	1.50	9.94	2.76	5.44	0.88	8.56
10 +	1.24	16.04	4.14	0.78	0.74	0.78	7.26	0.88	27.64	0.54
20 +	1.44	0.90	2.62	3.88	2.54	2.50	2.92	2.96	1.56	1.66
30 +	2.18	2.90	1.72	3.20	2.14					

对应于弱有效解，网页可达性和载入时间的倒数和网页访问率的对应关系分别如图4.14和图4.15所示。

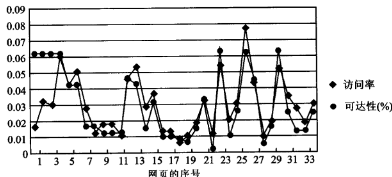


图4.14 网页可达性与访问率的对应关系（$cov=0.736$）

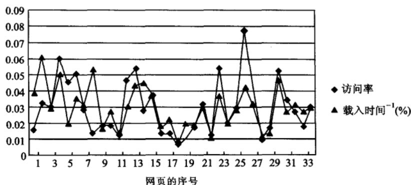


图 4.15 网页载入时间的倒数与访问率的对应关系（$cov = 0.672$）

优化前后附加链接的变化情况见表 4.6 所示，其中“+”和“-”分别表示增加和减少的链接。

表 4.4 不相关网页对

源网页	目标网页	源网页	目标网页
1	2,3,7,18,20,22,24,25,29,30,33	16	17,28,30
2	3,5,6,9,11,14,15,21,26,27,31,32	17	18,19,21,24,26,29,34
3	4,6,7,13,17,18,19,28,30	18	20,23,28,31
4	5,7,9,13,16,18,20,23,30,31,32,34	19	23,30,32
5	7,10,11,17,23,29,32,33	20	22,23,24,26,29,31
6	7,8,12,18,21,30,32	21	24,29,32
7	9,11,15,17,21,29,30,33	22	26,33,34
8	11,13,21,22,23,25,27	23	27
9	17,19,23,26,29,30,31,33,34	24	25,26,29
10	15,20,22,28,29,31,32	25	29,30,33
11	14,19,29,30	26	29,33,34
12	14,17,19,28	27	30,32,34
13	16,27	28	29,30,31,32
14	15,32	29	33,34
15	26,29,32	30	31,33,34

表 4.5 理想点和弱有效解

	初始解	理想点	弱有效解	与理想点之差
$f_1(x)$	0.381	0.796	0.736	0.061
$f_2(x)$	0.251	0.757	0.672	0.085*

注：“*”转化后单目标问题的最优解。

以上算法的平均运行时间为88.4 s，这在中、小型网站的优化中是可以接受的。考虑到Java的运行效率较低，而且网站的链接结构可以使用邻接表来存储，尚有提高算法运行效率的余地。

表4.6 优化前后附加链接的变化情况

+(0,30)	+(1,34)	+(2,13)	+(3,8)	+(3,21)	+(5,8)	+(6,1)	+(6,24)	+(7,12)
+(7,32)	+(8,0)	+(8,1)	+(10,9)	+(10,33)	+(11,17)	+(11,28)	+(12,2)	+(12,3)
+(13,25)	+(16,5)	+(16,6)	+(17,4)	+(17,6)	+(18,29)	+(20,5)	+(20,7)	+(21,4)
+(21,9)	+(22,29)	+(23,21)	+(24,7)	+(24,10)	+(25,11)	+(25,13)	+(26,12)	+(26,31)
+(27,11)	+(27,14)	+(28,15)	+(28,19)	+(29,16)	+(29,18)	+(30,15)	+(31,23)	+(31,26)
+(32,23)	+(32,26)	+(33,19)	+(33,24)	+(34,31)	+(34,32)	-(0,18)	-(2,17)	-(9,14)
-(23,30)	-(30,0)							

4.6.4 多盘刹车设计中的应用

1. 问题的描述

关于多盘刹车设计问题的详细描述参见本章参考文献［11］。这里直接给出如下多目标优化模型：

$$\min f_1(x) = 4.9\times10^{-5}(x_2^{\ 2}-x_1^{\ 2})(x_4-1)$$

$$\min f_2(x) = \frac{9.82\times10^{-6}(x_2^{\ 2}-x_2^{\ 2})}{x_3x_4(x_2^{\ 3}-x_1^{\ 3})}$$

$$\min f_3(x) = x_3$$

$$\text{s. t. } x_2 - x_1 - 20 \geqslant 0$$

$$30 - 2.5(x_4+1) \geqslant 0$$

$$0.4 - \frac{x_3}{3.14(x_2^{\ 2}-x_1^{\ 2})} \geqslant 0$$

$$1 - \frac{2.22\times10^{-3}x_3(x_2^{\ 3}-x_1^{\ 3})}{(x_2^{\ 2}-x_1^{\ 2})^2} \geqslant 0$$

$$\frac{2.66\times10^{-2}x_3x_4(x_2^{\ 3}-x_1^{\ 3})}{x_2^{\ 2}-x_1^{\ 2}} - 900 \geqslant 0$$

$$55 \leqslant x_1 \leqslant 80$$

$$75 \leqslant x_2 \leqslant 110$$

$$1\ 000 \leqslant x_3 \leqslant 3\ 000$$

$$2 \leqslant x_4 \leqslant 20$$

模型中包括4个决策变量，其中 x_1、x_2 和 x_3 3个为连续型，x_4 为离散型；3个目标函数，包括线性的和非线性的（高次）；除变量取值范围

之外，包括5个约束条件，其中包括高次的，这是一个混合多目标规划模型。

2. MOTS参数设置及运行环境

Baykasoglu（2005）使用4.6.2节中介绍的多目标禁忌搜索MOTS算法对该多目标模型进行求解，参数设置如下。

①邻域大小：20。

②邻域移动策略：简单策略。

③禁忌表长度：20。

④连续型变量步长：0.01。

⑤整数变量步长：1。

⑥最大迭代步数：20 000。

MOTS算法使用C ++语言实现，运行环境为Pentium Ⅳ个人计算机，CPU：1.60 Hz，内存：256 MB。

3. MOTS算法计算结果

按以上配置参数和运行环境，MOTS算法运行了大约10 min，求得5 964个Pareto最优解。在此之前，Osyczka和Kundu也求解了这个模型，使用简单随机搜索（Plain Stochastic）得到了19个Pareto最优解，使用遗传算法迭代20 000次得到了133个最优解。可见，相对于简单随机搜索和遗传算法，MOTS算法得到了相当多的高质量Pareto最优解。

同时，使用这三种方法求得的极限（Extreme）点，即单一目标函数的最优值，列出如表4.7所示，其中加粗部分对应得到优化的目标函数分量。可以看到，大部分情况下，MOTS算法的结果比其他两种方法好得多，只是$\min f_2(x)$对应情况稍差一点点。

表4.7 各方法求得极限点比较

求解方法	目标函数	$F(X)=[f_1(x), f_2(x), f_3(x)]^T$
一般随机搜索	$\min f_1(x)$	[**1.79**, 2.77, 2 920.9]
	$\min f_2(x)$	[3.76, **2.24**, 2 948.4]
	$\min f_3(x)$	[3.25, 2.80, **2 309.2**]
遗传算法	$\min f_1(x)$	[**1.66**, 2.87, 2 982.4]
	$\min f_2(x)$	[3.25, **2.11**, 2 988.3]
	$\min f_3(x)$	[3.91, 2.86, **2 255.1**]
MOTS算法	$\min f_1(x)$	[**0.131 156**, 41.353 2, 1 183.29]
	$\min f_2(x)$	[2.166 56, **2.15**, 2 981.64]
	$\min f_3(x)$	[1.153 09, 10.850 8, **1 000.03**]

本节给出了两个禁忌搜索算法的应用实例，4.6.3节的实例应用的是基本禁忌搜索算法，成功地求解了模型中有些变量不能解析化的模型，充分体现出禁忌搜索算法相对于传统算法的优越性。4.6.4节的实例为扩展后的应用于多目标的禁忌搜索MOTS算法，模型中包括连续变量和离散变量，目标函数与约束都是非线性的，搜索到的结果从各方面优越于文献中的其他方法。体现出禁忌搜索算法不但能求解单目标的组合优化，而且完全能求解连续的、多目标的优化问题。

问题与思考

1. 对于背包问题：7件财宝的价值为 a_i 和质量为 w_i（$i=1, \cdots, 7$）某人能背动的质量为120。设 $x_i=1$ 表示选择财宝 i，$x_i=0$ 表示不选择财宝 i。试用禁忌搜索算法求出最好解。初始解 $X=[1010101]$，邻域搜索选为加1减1运算，做5次迭代，禁忌长度取3（只用短期表）。

2. 某公司拟在4个地点建4个工厂。4个工厂的设计占地面积分别为 $R_1=9$，$R_2=8$，$R_3=4$，$R_4=5$；4个地点的地价分别为 $P_1=3$，$P_2=2$，$P_3=4$，$P_4=1$。公司的可用资金量为70。设状态 $X=[x_1, x_2, x_3, x_4]$，$x_i=k$ 表示工厂 i 选在地点 k，初始解为 $X=[1\ 3\ 2\ 4]$，用基本禁忌搜索作3次迭代，找出最优解，禁忌长度取3（只用短期表）。

3. 旅行商问题可简述如下：找一条经过 n 个城市的巡回（每个城市过且只过一次），极小化总路程。其中，城市 i、j 间的距离用 d_{ij} 表示。设计用禁忌搜索算法求解该问题的算法。要求写明编码方式、邻域选择策略、禁忌对象、渴望水平以及停止准则，并给出流程图。

4. 工作指派问题简述如下：n 个工作可以由 n 个工人分别完成。工人 i 完成工作 j 的时间为 d_{ij}。问如何安排可使总工作时间达到极小？试建立数学模型，并按禁忌搜索设计求解问题的算法（包括编码方式，邻域选择策略、禁忌对象、渴望水平、停止准则），并画出程序流程图。

5. 禁忌搜索算法与传统优化算法的最主要区别是什么？

6. 禁忌搜索算法与其他智能优化算法的最主要区别是什么？

7. 以旅行商问题为例，编写程序实现禁忌搜索算法，并体会禁忌表长度对算法性能的影响。你认为禁忌长度应该如何设置？

参考文献

[1] Battiti R, Tecchiolli G. The reactive tabu search [J]. ORSA Journal on Computing, 1994, 6 (2): 126-140.

[2] Baykasoglu A. Applying multiple objective tabu search to continuous

optimization problems with a simple neighbourhood strategy [J]. Int. J. Numer. Meth. Engng, 2006, 65 (3): 406 - 424.

[3] Ehrgott M, Gandibleux X. Approximate solution methods for multi - objective combinatorial optimization [J]. Top, 2004, 12: 1 - 89.

[4] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job shop problem [J]. Management Science, 1996, 42 (6): 797 - 813.

[5] Glover Fred, Hanafi Saïd. Tabu search and finite convergence [J]. Discrete Applied Mathematics, 2002, 119: 3 - 36.

[6] Glover F, Kelly J P, Laguna M. Genetic algorithms and tabu search: hybrids for optimization [J]. Computers Ops. Res., 1995, 22 (1): 111 - 134.

[7] Glover F, Taillard E, Werra D de. A user's guide to tabu search [J]. Annals of Operations Research, 1993, 41 (1 - 4): 3 - 28.

[8] Glover F. Heuristics for integer programming using surrogate constraints [J]. Decision Science, 1977, 8 (1): 156 - 166.

[9] Glover F. Tabu search - Part I [J]. ORSA Journal on Computing, 1989, 1 (3): 190 - 206.

[10] Glover F. Tabu search - Part II [J]. ORSA Journal on Computing, 1990, 2 (1): 4 - 32.

[11] Osyczka A, Kundu S. A modified distance method for multi - criteria optimization using genetic algorithms [J]. Computers and Industrial Engineering, 1996, 30: 871 - 882.

[12] Reeves C. Genetic algorithms and neighborhood search [J]. In Gogarty, 128: 115 - 130.

[13] Rosenkrantz D, Stearns R, Lewis P. An analysis of several heuristics for the traveling salesman problem [J]. SLAM Journal of Computing, 1977, 6 (1): 563 - 581.

[14] Salhi S. Defining tabu list size and aspiration criterion within tabu search methods [J]. Computers & Operations Research, 2002, 29 (1): 67 - 86.

[15] Talbi E G, Hafidi Z, Geib J M. A parallel adaptive tabu search approach [J]. Parallel Computing, 1998, 24 (14): 2003 - 2019.

[16] Wang Y, Wang D, Ip W H. Optimal design of link structure for e - supermarket website [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2006, 36 (2):

338 - 355.

[17] 蔡砥，藤丽，王铮. 一种禁忌搜索算法在计算网格中的并行化策略 [J]. 微电子学与计算机，2004，21 (6)：115 - 119.

[18] 曹立斌，周建兰. 一种改进的禁忌搜索法在函数优化问题中的应用 [J]. 微机发展，2003，13：39 - 42.

[19] 董宏光，秦立民，王涛，樊栓狮，姚平经. 基于自适应并行禁忌搜索的精馏分离序列优化综合 [J]. 化工学报，2004，55 (10)：1669 - 1673.

[20] 方永慧，刘光远，贺一，邱玉辉. 一种基于插入法的禁忌搜索算法 [J]. 西南师范大学学报 (自然科学版)，2003，28 (6)：887 - 891.

[21] 郭宇，陈立平，王书亭，钟毅方. 一种面向车间物流模型的仿真优化方法 [J]. 小型微型计算机系统，2003，24 (12)：2316 - 2320.

[22] 贺一. 基于禁忌搜索的前向神经网络在函数逼近中的应用 [J]. 西南师范大学学报 (自然科学版)，2004，29 (3)：361 - 365.

[23] 贾永基，谷寒雨，席裕庚. 一类货运车辆调度问题的混合禁忌搜索算法 [J]. 信息与控制，2004，33 (6)：724 - 728.

[24] 刘江华，陈佳品，程君实. 基于一种改进禁忌搜索算法优化离散隐马尔可夫模型 [J]. 计算机工程与应用，2003，(20)：92 - 94.

[25] 施文俊，何小荣，陈丙珍，邱彤. TS 法的改进及其在求解化工优化问题中的应用 [J]. 化工学报，2004，55 (10)：1665 - 1668.

[26] 唐普英，李志辉，黄顺吉. 基于遗传算法和禁忌搜索的多用户检测器 [J]. 电子科技大学学报，2004，33 (5)：499 - 502.

[27] 童刚，李光权，刘宝坤. 一种用于 Job - Shop 调度问题的改进禁忌搜索算法 [J]. 系统工程理论与实践，2001，(9)：48 - 52.

[28] 王凌. 智能优化算法及其应用 [M]. 北京：清华大学出版社，2001.

[29] 王有为，汪定伟. 电子超市网站链接结构优化的多目标模型 [J]. 控制理论与应用，2004，21 (1)：6 - 10.

[30] 王有为. 电子商务网站链接结构优化方法的研究 [D]. 沈阳：东北大学信息科学与工程学院，2003.

[31] 吴璟莉，李陶深. 遗传算法与禁忌搜索算法的混合策略在 VRPTM 问题上的应用 [J]. 计算机工程与应用，2004，(18)：

54 - 57.

[32] 阎志伟，牛轶峰，李汉铃. 基于并行禁忌遗传算法 PTGA 的预警卫星传感器调度研究 [J]. 宇航学报，2003，24 (6)：598 - 603.

[33] 杨银国，张伏生，贺春光，王春娟，李宁. 基于主动禁忌搜索的配电网无功电压优化控制 [J]. 西安交通大学学报，2005，39 (8)：38 - 42.

[34] 杨振野，赖强，李豪彦. 自调整禁忌搜索算法及其在无损检测中的应用 [J]. 仪器仪表学报，2004，25 (3)：317 - 320.

[35] 衣杨，汪定伟. TS 求解多机成组工件调度 [J]. 东北大学学报 (自然科学版)，2001，22 (2)：188 - 191.

[36] 衣杨，汪定伟. 并行多机成组工件调度的禁忌搜索方法 [J]. 系统工程，2000，18 (6)：11 - 17.

[37] 张颖，刘艳秋. 软计算方法 [M]. 北京：科学出版社，2002.

[38] 周创明，华继学，李成海. 具有禁忌算子的遗传算法目标优化分配 [J]. 空军工程大学学报 (自然科学版)，2005，6 (2)：87 - 91.

第 5 章　模拟退火算法

模拟退火（Simulated Annealing，SA）算法是一种通用的随机搜索算法，是对局部搜索算法的扩展。与一般局部搜索算法不同，SA 以一定的概率选择邻域中目标值相对较小的状态，是一种理论上的全局最优算法。模拟退火算法是源于对热力学中退火过程的模拟，在某一给定初温下，通过缓慢下降温度参数，使算法能够在多项式时间内给出一个近似最优解。虽然早在 1953 年，Metropolis 就提出了模拟退火算法的思想，但直到 1983 年，Kirkpatrick 成功地将 SA 应用在组合最优化问题中，才真正创建了现代的模拟退火算法。本章将对模拟退化算法的基本思想、算法的构造、实现技术、收敛性分析以及实际应用一一介绍。

5.1 导　　言

早在 1953 年，Metropolis 等人就提出了原始的 SA 算法，但是并没有引起反响，直到 1983 年，Kirkpatrick 等人提出了现代的 SA 算法，并成功地利用它来解决大规模的组合最优化问题。由于现代 SA 算法能够有效地解决具有 NP 复杂性的问题，避免陷入局优，克服初值依赖性等优点，目前已在工程中得到了广泛的应用，诸如 VLS、生产调度、控制工程、机器学习、神经网络、图像处理等领域。模拟退火算法的基本思想是源于热力学中的退火过程，因此首先介绍一下热力学当中的退火过程。

5.1.1　热力学中的退火过程

金属物体被加热到一定温度后，它的所有分子在状态空间自由运动，随着温度的逐渐下降，分子停留在不同的状态，分子运动逐渐趋于有序，最后以一定的结构排列。这种由高温向低温逐渐降温的热处理过程就称为退火。退火是一种物理过程，在退火过程中系统的熵值不断减小，系统能量随温度的降低趋于最小值，也就是说，金属物体从高能状态转移到低能状态，变得较为柔韧。一个退火过程一般由以下三部分组成。

1. 加温过程

其目的是增强分子的热运动，使其偏离平衡位置。当温度足够高

时，固体将溶解为液体，分子的分布从有序的结晶态转变成无序的液态，从而消除系统原先可能存在的非均匀态，使随后进行的冷却过程以某一平衡态为起点。溶解过程与系统的熵增过程相联系，系统能量也随温度的升高而增大。

2. 等温过程

这个过程是为了保证系统在每一个温度下都达到平衡态，最终达到固体的基态。根据热平衡封闭系统的热力学定律—自由能减少定律："对于与周围环境交换热量而温度不变的封闭系统，系统状态的自发变化总是朝自由能减少的方向进行，当自由能达到最小时，系统就达到了平衡态。"

3. 冷却过程

其目的是使分子的热运动减弱并渐趋有序，系统能量逐渐下降，当温度降至结晶温度后，分子运动变成了围绕晶体格点的微小振动，液体凝固成固体的晶态，从而得到低能的晶体结构。

金属物体被加热到一定温度后，若急剧降低温度，则物体只能冷凝为非均匀的亚稳态，这就是热处理过程中的淬火效应。淬火也是一种物理过程，由于物体在这个过程中并没有达到平衡态，所以系统能量并不会达到最小值，也就是说，金属依然保持在高能状态，虽然能提高其强度和硬度，但韧性会减弱。

退火

金属： 高温 —缓慢下降→ 低温

高能状态 ———→ 低能状态

淬火

金属： 高温 —快速下降→ 低温

高能状态 ———→ 低能状态

图 5.1 退火与淬火过程

5.1.2 退火与模拟退火

金属物体的退火过程实际上就是随温度的缓慢降低，金属由高能无序的状态转变为低能有序的固体晶态的过程。在退火中，需要保证系统在每一个恒定温度下都要达到充分的热平衡，这个过程可以用 Monte Carlo 的方法加以模拟，该方法虽然比较简单，但需要大量采样才能获得比较精确的结果，计算量较大。鉴于物理系统倾向于能量较低的状态，而热运动又妨碍它准确落到最低态的物理形态，采样时只需着重取那些有贡献作用的状态则可较快达到较好的结果。1953 年，Metropolis

等提出了一种重要性采样法，即以概率来接受新状态。具体而言，在温度 t，由当前状态 i 产生新状态 j，两者的能量分别为 E_i 和 E_j，若 $E_i > E_j$，则接受新状态 j 为当前状态；否则，以一定的概率 $p_r = \exp\left[\frac{-(E_j - E_i)}{kt}\right]$ 来接受状态 j，其中 k 为 Boltzmann 常数。这里，$\exp[x]$ 即为指数函数 e^x。当这种过程多次重复，即经过大量迁移后，系统将趋于能量较低的平衡态，各状态的概率分布将趋于一定的正则分布。这种接受新状态的方法被称为 Metropolis 准则，它能够大大减少采样的计算量。

对于一个典型的组合优化问题，其目标是寻找一个 x^*，使得对于 $\forall x_i \in \boldsymbol{\Omega}$，存在 $c(x^*) = \min c(x_i)$，其中 $\boldsymbol{\Omega} = \{x_1, x_2, \cdots, x_n\}$ 为由所有解构成的解空间，$c(x_i)$ 为解 x_i 对应的目标函数值。利用简单的爬山算法来求解这类优化问题时，在搜索过程中很容易陷入局优点，具有相当的初值依赖性。Kirkpatrick 等人根据金属物体的退火过程与组合优化问题之间存在的相似性，并且在优化过程中采用 Metropolis 准则作为搜索策略，以避免陷入局部最优，并最终趋于问题的全局最优解。

在 SA 中，优化问题中的一个解 x_i 及其目标函数 $c(x_i)$ 分别可以看成物理退火中物体的一个状态和能量函数，而最优解 x^* 就是最低能量的状态。而设定一个初始高温、基于 Metropolis 准则的搜索和控制温度参数 t 的下降分别相当于物理退火的加温、等温和冷却过程。表 5.1 就描述了一个组合优化问题的求解过程与物理退火过程之间的对应关系。

表 5.1 组合优化问题的求解与物理退火

优化问题	物理退火
解	状态
目标函数	能量函数
最优解	最低能量的状态
设定初始高温	加温过程
基于 Metropolis 准则的搜索	等温过程
温度参数 t 的下降	冷却过程

5.2 退火过程的数学描述和 Boltzmann 方程

前节指出，退火过程是一个变温物体缓慢降温从而达到分子间能量最低状态的过程。设热力学系统 S 中有 n 个状态，注意这里的状态数是

有限且离散的，其中状态 i 的能量为 E_i。在温度 T_k 下，经一段时间达到热平衡，这时处于状态 i 的概率为

$$P_i(T_k) = C_k \exp\left(\frac{-E_i}{T_k}\right) \tag{5.1}$$

式中，C_k是一个参数，能够根据已知条件计算获得。由于 S 中一共存在 n 个状态，故在温度 T_k下，S 必然处于其中的一个状态，也就是说

$$\sum_{j=1}^{n} P_j(T_k) = 1 \tag{5.2}$$

代入式（5.1），则

$$\sum_{j=1}^{n} C_k \exp\left(\frac{-E_j}{T_k}\right) = 1 \Rightarrow C_k \sum_{j=1}^{n} \exp\left(\frac{-E_j}{T_k}\right) = 1$$

由此得到待定系数

$$C_k = \frac{1}{\sum_{j=1}^{n} \exp\left(\frac{-E_j}{T_k}\right)}$$

于是，式（5.1）可以表示为

$$P_i(T_k) = \frac{\exp\left(\frac{-E_i}{T_k}\right)}{\sum_{j=1}^{n} \exp\left(\frac{-E_j}{T_k}\right)} \tag{5.3}$$

根据式（5.1），对于任意两个能量状态 E_1 和 E_2，若存在 $E_1 < E_2$，则在同一个温度 T_d下，有

$$\frac{P_1(T_k)}{P_2(T_k)} = \frac{C_k \exp\left(\frac{-E_1}{T_k}\right)}{C_k \exp\left(\frac{-E_2}{T_k}\right)} = \exp\left(-\frac{E_2 - E_1}{T_k}\right)$$

因为 $E_2 - E_1 > 0$，有

$$\exp\left(-\frac{E_2 - E_1}{T_k}\right) < 1, \quad \forall T_k > 0$$

所以必有

$$P_1(T_k) > P_2(T_k), \quad \forall T_k > 0 \tag{5.4}$$

这表明在同一温度下，式（5.4）表示 S 处于能量小的状态的概率比处于能量大的状态的概率要大，也就是说，在同一温度下，随着状态能量函数的减小，其概率将会增大，两者之间存在着反向变化的关系。

式（5.1）和式（5.3）又称为 Boltzmann 方程，用于描述系统 S 在给定温度下，处于某一状态的概率分布。根据 Boltzmann 方程来分析状

态概率随温度变化的规律。对 $P_i(T_k)$ 求对温度的导数，得到

$$\frac{\partial P_i(T_k)}{\partial T_k}=\frac{\partial\left[\exp\left(\frac{-E_i}{T_k}\right)\Big/\sum_{i=1}^{n}\exp\left(\frac{-E_i}{T_k}\right)\right]}{\partial T_k}$$

$$=\frac{\exp\left(\frac{-E_i}{T_k}\right)\cdot\frac{E_i}{T_k^2}\cdot\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)-\exp\left(\frac{-E_i}{T_k}\right)\cdot\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)\cdot\frac{E_j}{T_k^2}}{\left[\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)\right]^2}$$

$$=\frac{\exp\left(\frac{-E_i}{T_k}\right)}{T_k^2\cdot\left[\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)\right]^2}\left[\sum_{j=1}^{n}(E_i-E_j)\cdot\exp\left(\frac{-E_j}{T_k}\right)\right]\tag{5.5}$$

设 i^* 为 S 中最低能量的状态，则 $\forall j$，存在 $E_{i^*}-E_j\leqslant 0$，而

$$\frac{\exp\left(\frac{-E_i}{T_k}\right)}{T_k^2\cdot\left[\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)\right]^2}>0,\exp\left(\frac{-E_j}{T_k}\right)>0$$

故有

$$\frac{\partial P_{i^*}(T_k)}{\partial T_k}<0,\qquad\forall T_k$$

因此，$P_{i^*}(T_k)$ 关于温度 T_k 是单调递减的。又有

$$P_{i^*}(T_k)=\frac{\exp\left(\frac{-E_{i^*}}{T_k}\right)}{\sum_{j=1}^{n}\exp\left(\frac{-E_j}{T_k}\right)}$$

$$=\frac{1}{\sum_{j=1}^{n}\exp\left[\frac{-(E_j-E_{i^*})}{T_k}\right]}\tag{5.6}$$

接着，分两种情况来讨论当 $T_k\to 0$ 时，如何计算 $P_{i^*}(T_k)$。

（1）当 S 中仅存在一个最低能量状态 i^* 时，也就是说，在解空间中存在唯一的全局最优解时，则当 $T_k\to 0$ 时，对于 $\forall j\neq i^*$，存在

$$E_j-E_{i^*}>0\Rightarrow\frac{-(E_j-E_{i^*})}{T_k}\to-\infty\Rightarrow\exp\left[\frac{-(E_j-E_{i^*})}{T_k}\right]=0$$

所以

$$P_{i^*}(T_k) = \frac{1}{\sum_{j=1}^{n} \exp\left[\frac{-(E_j - E_{i^*})}{T_k}\right]}$$

$$= \frac{1}{\exp\left[\frac{-(E_{i^*} - E_{i^*})}{T_k}\right]} = 1 \tag{5.7}$$

（2）当 S 中存在 n_0 个最低能量状态时，假设 i^* 是其中的一个状态，这相当于，在解空间中存在若干个全局最优解。根据上面的推导，能够获得，当 $T_k \to 0$ 时

$$P_{i^*}(T_k) = \frac{1}{\sum_{j=1}^{n} \exp\left[\frac{-(E_j - E_{i^*})}{T_k}\right]} = \frac{1}{n_0} \tag{5.8}$$

可见，当 $T_k \to 0$ 时，S 处于 i^* 状态的概率是 $1/n_0$。由于 S 中存在 n_0 个最低能量状态，所以，当 $T_k \to 0$ 时，S 处于最低能量状态的概率趋向 1。

因此，根据式（5.7）和式（5.8）可知，当 $T_k \to 0$ 时 S 处于最低能量状态的概率趋向于 1。

用 $\overline{E}(T_k)$ 来表示温度 T_k 下的平均能量，则

$$\overline{E}(T_k) = \sum_{j=1}^{n} E_j \cdot P_j(T_k)$$

由式（5.7）和式（5.8），易知当 $T_k \to 0$ 时，$\overline{E}(T_k) \to E_{i^*}$。

根据式（5.1）可知，当温度 T_k 很大时，$\frac{E_i}{T_k} \to 0$，此时 $P_i(T_k) \approx \frac{1}{n}$。也就是说，当温度很高时，$S$ 处于各状态的概率几乎相等。SA 开始做广域的随机搜索，随着温度的下降，状态概率 $P_i(T_k)$ 的差别开始扩大，当 $T_k \to 0$ 时，$\frac{E_i}{T_k} \to \infty$，此时 E_i 与 E_j 之间的微小差别将会引起 $P_i(T_k)$ 和 $P_j(T_k)$ 的剧烈变化。

例如：假设 $E_i = 90$，$E_j = 100$，当温度 $T_k = 100$ 时

$$\frac{P_i(T_k)}{P_j(T_k)} = \frac{C_k \cdot e^{-\frac{90}{100}}}{C_k \cdot e^{-\frac{100}{100}}} = \frac{0.406C_k}{0.367C_k} = \frac{0.406}{0.367} \approx 1$$

此时，$P_i(T_k) \approx P_j(T_k)$。

当温度 $T_k = 1$ 时

$$\frac{P_i(T_k)}{P_j(T_k)} = \frac{C_k \cdot e^{-\frac{90}{1}}}{C_k \cdot e^{-\frac{100}{1}}} = \frac{8.194 \times 10^{-40} C_k}{3.72 \times 10^{-44} C_k} \approx 20\ 000$$

此时，$P_i(T_k)+P_j(T_k)\approx P_i(T_k)$。

在上面的小例子中，当温度 T_k等于100时，S处于低能状态 E_i和处于高能状态 E_j的概率几乎相等；而当温度 T_k减小到1的时候，S处于低能状态 E_i的概率比处于高能状态 E_j的概率大约高20 000倍。由此可以看出，在高温下，S可以处于任何能量状态，此时SA可以看成是在进行广域搜索，以避免陷入局优；在低温下，S只能处于能量较小的状态，此时SA可以看成是在做局域搜索，以便于将解精细化；当温度无限趋近于零时，S只能处于最小能量的状态，此时SA就获得了全局最优解，同时这个小例子也进一步验证了当 $T_k\to 0$ 时，S会以概率1趋于最低能量状态。

5.3 模拟退火算法的构造及流程

SA算法是一种启发式的随机寻优算法，它模拟了物理退火过程，由一个给定的初始高温开始，利用具有概率突跳特性的Metropolis抽样策略在解空间中随机进行搜索，伴随温度的不断下降重复抽样过程，最终得到问题的全局最优解，这就是SA算法的基本思想。

5.3.1 算法的要素构成

在SA算法执行过程中，算法的效果取决于一组控制参数的选择，关键技术的设计对算法性能影响很大。本节从算法使用的角度讨论算法实现中的一些要素，包括状态表达，邻域定义，热平衡达到和降温控制等的概念。

1. 状态表达

同前面几章介绍的遗传算法（GA）和禁忌搜索（TS）中的编码含义相同，状态表达是利用一种数学形式来描述系统所处的一种能量状态。在SA中，一个状态就是问题的一个解，而问题的目标函数就对应于状态的能量函数。常见的状态表达方法有适用于背包问题和指派问题的 **0-1** 编码表示法，适用于TSP问题和调度问题的自然数编码表示法以及适用于各种连续函数优化的实数编码表示法等。状态表达是SA的基础工作，直接决定着邻域的构造和大小，一个合理的状态表达方法会大大减小计算复杂性，改善算法的性能。

2. 邻域定义与移动

同TS一样，SA也是基于邻域搜索的。邻域定义的出发点应该是保证其中的解能尽量遍布整个解空间，其定义方式通常是由问题的性质所

决定的，在前文介绍的 TS 中已经对邻域的定义方法做了阐述，这里就不再对它加以讨论了。

给定一个解的邻域之后，接下来就要确定从当前解向其邻域中的一个新解进行移动的方法。SA 算法采用了一种特殊的 Metropolis 准则的邻域移动方法，也就是说，依据一定的概率来决定当前解是否移向新解。在 SA 中，邻域移动分为两种方式：无条件移动和有条件移动。若新解的目标函数值小于当前解的目标函数值（新状态的能量小于当前状态的能量），则进行无条件移动；否则，依据一定的概率进行有条件移动。

设 i 为当前解，j 为其邻域中的一个解，它们的目标函数值分别为 $f(i)$ 和 $f(j)$，用 Δf 来表示它们的目标值增量，$\Delta f = f(j) - f(i)$。

若 $\Delta f < 0$，则算法无条件从 i 移动到 j（此时 j 比 i 好）；

若 $\Delta f > 0$，则算法依据概率 p_{ij} 来决定是否从 i 移向 j（此时 i 比 j 好），这里 $p_{ij} = \exp\left(\frac{-\Delta f}{T_k}\right)$，其中 T_k 是当前的温度。

这种邻域移动方式的引入是实现 SA 进行全局搜索的关键因素，能够保证算法具有跳出局部最小和趋向全局最优的能力。当 T_k 很大时，p_{ij} 趋近于 1，此时 SA 正在进行广域搜索，它会接受当前邻域中的任何解，即使这个解要比当前解差。而当 T_k 很小时，p_{ij} 趋近于 0，此时 SA 进行的是局域搜索，它仅会接受当前邻域中更好的解。

3. 热平衡达到

热平衡的达到相当于物理退火中的等温过程，是指在一个给定温度 T_k 下，SA 基于 Metropolis 准则进行随机搜索，最终达到一种平衡状态的过程。这是 SA 算法中的内循环过程，为了保证能够达到平衡状态，内循环次数要足够大才行。但是在实际应用中达到理论的平衡状态是不可能的，只能接近这一结果。最常见的方法就是将内循环次数设成一个常数，在每一温度，内循环迭代相同的次数。次数的选取同问题的实际规模有关，往往根据一些经验公式获得。此外，还有其他一些设置内循环次数的方法，比如根据温度 T_k 来计算内循环次数，当 T_k 较大时，内循环次数较少，当 T_k 减小时，内循环次数增加。

4. 降温函数

降温函数用来控制温度的下降方式，这是 SA 算法中的外循环过程。利用温度的下降来控制算法的迭代是 SA 的特点，从理论上说，SA 仅要求温度最终趋于 0，而对温度的下降速度并没有什么限制，但这并不意味着可以随意下降温度。由于温度的大小决定着 SA 进行广域搜索

还是局域搜索，当温度很高时，当前邻域中几乎所有的解都会被接受，SA进行广域搜索；当温度变低时，当前领域中越来越多的解将被拒绝，SA进行领域搜索。若温度下降得过快，SA将很快从广域搜索转变为局域搜索，这就很可能造成过早地陷入局部最优状态，为了跳出局优，只能通过增加内循环次数来实现，这就会大大增加算法进程的CPU时间。当然，如果温度下降得过慢，虽然可以减少内循环次数，但是由于外循环次数的增加，也会影响算法进程的CPU时间。可见，选择合理的降温函数能够帮助提高SA算法的性能。

常用的降温函数有两种：

(1) $T_{k+1}=T_k \cdot r$，其中$r\in(0.95, 0.99)$，r越大温度下降得越慢。这种方法的优点是简单易行，温度每一步都以相同的比率下降。

(2) $T_{k+1}=T_k-\Delta T$，ΔT是温度每一步下降的长度。这种方法的优点是易于操作，而且可以简单控制温度下降的总步数，温度每一步下降的大小都相等。

此外，初始温度和终止温度的选择对SA算法的性能也会有很大的影响。一般来说，初始温度T_0要足够大，也就是使

$$\frac{f_i}{T_0}\approx 0$$

以保证SA在开始时能够处在一种平衡状态。在实际应用中，要根据以往经验，通过反复实验来确定T_0的值。而终止温度T_f要足够小，以保证算法有足够的时间获得最优解。T_f的大小一般可以根据降温函数的形式来确定，若降温函数为$T_{k+1}=T_k \cdot r$，则可以将T_f设成一个很小的正数；若$T_{k+1}=T_k-\Delta T$，则可以根据预先设定的外循环次数和初始温度T_0计算出终止温度T_f的值。

5.3.2 算法的计算步骤和流程图

一个优化问题可以描述为

$$\min f(i),\ i\in S$$

其中，S是一个离散有限状态空间，i代表状态。针对这样一个优化问题，SA算法的计算步骤能够描述如下：

第1步：初始化，任选初始解$i\in S$，给定初始温度T_0和终止温度T_f，令迭代指标$k=0$，$T_k=T_0$。

第2步：随机产生一个邻域解$j\in N(i)$（$N(i)$表示i的邻域），计算目标值增量$\Delta f=f(j)-f(i)$。

第3步：若$\Delta f<0$，令$i=j$转第4步；否则产生$\xi=U(0, 1)$，若

$\exp\left(-\frac{\Delta f}{T_k}\right) > \xi$，则令 $i=j$。

第 4 步：若达到热平衡（内循环次数大于 $n(T_k)$）转第 5 步；否则转第 2 步。

第 5 步：降低 T_k，$k=k+1$，若 $T_k < T_f$，则算法停止，否则转第 2 步。

根据上述计算步骤，SA 的流程图能够表示为如图 5.2 所示。

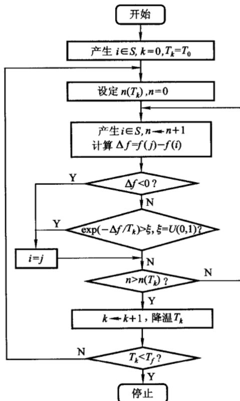


图 5.2　模拟退火算法流程图

5.3.3　一个简单的算例

下面以一个简单的例子来说明 SA 是怎么工作的。

1. 问题提出

以一个简单的单机极小化总流水时间的排序问题为例：设有四个工件需要在一台机床上加工，$P_1=8$，$P_2=18$，$P_3=5$，$P_4=15$ 分别是这

四个单道工序工件在机床上的加工时间，问应如何在这个机床上安排各工件加工的顺序，使工件加工的总流水时间最小？

2. 预备知识

工件的流水时间（Flowtime）即工件在系统中的总逗留时间，流水时间用于度量系统对各个服务需求的反应，表示工件在到达和离开系统的时间长度。在本例中，令所有工件都在时间 0 到达，则工件的流水时间就等于它的完工时间。

对于这种总流水时间问题可以用工件加工时间的非减顺序来调度工件，也就是著名的最短加工时间调度规则（SPT 规则）。那么按 SPT 规则，本例中保证总流水时间最小的最优的工件加工顺序为 3—1—4—2，它的总流水时间 F 可以计算如下

$$F_{[1]}=P_{[1]}$$

$$F_{[2]}=P_{[1]}+P_{[2]}$$

$$F_{[3]}=P_{[1]}+P_{[2]}+P_{[3]}$$

$$F_{[4]}=P_{[1]}+P_{[2]}+P_{[3]}+P_{[4]}$$

于是，$F=[P_{[1]},\ P_{[2]},\ P_{[3]},\ P_{[4]}]\begin{bmatrix}4\\3\\2\\1\end{bmatrix}=4\times5+3\times8+2\times15+1\times18=92$

注释：式中 $[k]$ 表示排在第 k 位的工件的标号。如：工件 3 排在第 1 位，则 $[1]=3$。

3. 用 SA 求解这个问题

（1）状态表达

本例中状态采用一种顺序编码来表达，其中工件是按加工的顺序排列的。例如，对于一个工件的加工顺序

2—4—1—3

可简单的表示为

[2 4 1 3]

（2）邻域定义

对于采用顺序编码的状态表达方法来说，最自然的邻域可以定义为工件顺序中两两换位的集合。例如，从一个当前状态

[2 4 1 3]

对其中的两个工件进行换位（2 与 1 进行换位），则获得了一个新状态

[1 4 2 3]

这样就完成了一次邻域移动。

(3) 温度参数设置

设初始温度 $T_0=100$，终止温度 $T_f=60$。

降温函数定义为 $T_{k+1}=T_k-\Delta T$，其中 $\Delta T=20$。

热平衡达到

通过设置内循环的迭代次数 $n(T_k)$ 来实现热平衡，这里设 $n(T_k)=3$。

4. SA 的求解过程

随机产生一个初始解 $\boldsymbol{i}=[1\ 4\ 2\ 3]$，其目标函数值 $f(\boldsymbol{i})=118$，SA 开始运行。

(1) 当前温度 $T_k=100$，进入内循环，令内循环次数 $n=0$。

①内循环次数 n 加 1，随机产生一个邻域解 $\boldsymbol{j}=[1\ 3\ 2\ 4]$（此时 4 和 3 换位），其目标函数值 $f(\boldsymbol{j})=98$；计算目标函数值增量 $\Delta f=-20$，由于 $\Delta f<0$，故进行无条件转移，令 $\boldsymbol{i}=\boldsymbol{j}$。

②$n\leftarrow n+1$，$\boldsymbol{j}=[4\ 3\ 2\ 1]$，$f(\boldsymbol{j})=119$；计算 $\Delta f=21$，由于 $\Delta f>0$，故进行有条件转移，计算

$$e^{-\frac{\Delta f}{T_k}}=0.8106$$

随机产生 $\xi\in U(0,1)$，$\xi=0.7414$，因为

$$e^{-\frac{\Delta f}{T_k}}=0.8106>0.7414(\xi)$$

所以进行邻域移动，令 $\boldsymbol{i}=\boldsymbol{j}$。

③$n\leftarrow n+1$，$\boldsymbol{j}=[4\ 2\ 3\ 1]$，$f(\boldsymbol{j})=132$，因为

$$e^{-\frac{\Delta f}{T_k}}=0.8781>0.3991\ (\xi)$$

所以进行有条件转移，令 $\boldsymbol{i}=\boldsymbol{j}$。

注释：在②③中，虽然目标值变大，但是搜索范围变大。

(2) 降低温度，$T_k=100-20=80$，$n=0$。

①$n\leftarrow n+1$，$\boldsymbol{j}=[4\ 2\ 1\ 3]$，$f(\boldsymbol{j})=135$，因为

$$e^{-\frac{\Delta f}{T_k}}=0.9632>0.3413(\xi)$$

所以进行邻域移动，令 $\boldsymbol{i}=\boldsymbol{j}$。

②$n\leftarrow n+1$，$\boldsymbol{j}=[4\ 3\ 1\ 2]$，$f(\boldsymbol{j})=109$，因为

$$\Delta f=-26<0$$

进行无条件转移，令 $\boldsymbol{i}=\boldsymbol{j}$。

③$n\leftarrow n+1$，$\boldsymbol{j}=[4\ 3\ 2\ 1]$，$f(\boldsymbol{j})=119$，因为

$$e^{-\frac{\Delta f}{T_k}}=0.8825>0.9286(\xi)$$

所以，不进行邻域移动，令 $i=i$。

注释：在③中，由于产生的伪随机数 ξ 大于转移概率 $e^{-\frac{\Delta f}{T_k}}$，所以系统会停留在 4—3—1—2 状态，目标值仍然为 109。

（3）降低温度，$T_k=80-20=60$，$n=0$

① $n \leftarrow n+1$，$j=[1\ 3\ 4\ 2]$，$f(j)=95$，因为

$$\Delta f=-24<0$$

所以进行无条件转移，令 $i=j$。

② $n \leftarrow n+1$，$j=[3\ 1\ 4\ 2]$，$f(j)=92$，因为

$$\Delta f=-3<0$$

进行无条件转移，令 $i=j$。

③ $n \leftarrow n+1$，$j=[2\ 1\ 4\ 3]$，$f(j)=131$，因为

$$e^{-\frac{\Delta f}{T_k}}=0.5220>0.7105\ (\xi)$$

所以，不进行邻域移动，令 $i=i$。

SA 停止运行，输出最终解 $i=[3\ 1\ 4\ 2]$，也就是说，终止于状态 3—1—4—2，目标函数值为 92。

虽然在这个简单的算例中，SA 终止于最优解，但是在实际应用过程中想做到这一点，就必须在算法设计过程中同时满足下列条件：

① 初始温度足够高。

② 热平衡时间足够长。

③ 终止温度足够低。

④ 降温过程足够慢。

以上条件在实际应用过程中很难同时得到满足，而且 SA 会接受性能较差的解，所以其最终解有可能比运算过程中遇到的最好解性能差，因此在 SA 运行过程中常常要记录遇到的最好可行解（历史最优解），当算法停止时，输出这个历史最优解。

5.4 算法的收敛性分析

与前文介绍的遗传算法 GA 和禁忌搜索 TS 相比，SA 的一大优点是理论较为完善，下面就基于 Markov 过程来分析 SA 算法的收敛性。

5.4.1 Markov 过程

Markov 链最初是由 Markov 于 1906 年研究而得名，至今它的理论已发展的较为深入和系统，在自然科学、工程技术及经济管理等各个领域

得到广泛的应用，本节将利用 Markov 链作为描述和分析 SA 算法的数学工具，首先介绍一些基本的概念。

1. Markov 链

状态：表示每个时刻开始处于系统中的一种特定自然状况或客观条件的表达，它描述了研究问题过程的状况。描述状态的变量称为状态变量，可用一个数，一组数或一向量（多维情况）来描述。

状态转移概率：表示在某一时刻从状态 i 转移到状态 j 的可能性。

无后效性：如果在某时刻状态给定后，则在这时刻以后过程的发展不受这时刻以前各段状态的影响。换句话说，达到一个状态后，决策只与当前状态有关，而与以前的历史状态无关，当前的状态是以往历史的一个总结。这个性质就称为无后效性。

根据上述概念，令离散参数 $T=\{0,1,2,\cdots\}=N_0$，状态空间 $S=\{0,1,2,\cdots\}$，如果随机序列 $\{X_n, n\geqslant 0\}$ 对于任意 $i_0, i_1, \cdots, i_n, i_{n+1}\in S$，$n\in N_0$ 及

$$P\ \{X_0=i_0, X_1=i_1, \cdots, X_n=i\}>0$$

存在

$$P\{X_{n+1}=i_{n+1}\mid X_0=i_0, X_1=i_1, \cdots, X_n=i_n\}$$
$$=P\{X_{n+1}=i_{n+1}\mid X_n=i_n\} \tag{5.9}$$

则称其为 Markov 链。式（5.9）刻画了 Markov 链的无后效性，若 S 有限，则称为有限状态 Markov 链。

对于 $\forall i, j\in S$，称

$$P\{X_{n+1}=j\mid X_n=i\}\triangleq p_{ij}(n) \tag{5.10}$$

为 n 时刻的一步转移概率。

若对于 $\forall i, j\in S$，存在

$$p_{ij}(n)\equiv p_{ij} \tag{5.11}$$

即 p_{ij}与 n 无关，则称 $\{X_n, n\geqslant 0\}$ 为齐次 Markov 链。记 $\boldsymbol{P}=(p_{ij})$，称 $\boldsymbol{P}$ 为 $\{X_n, n\geqslant 0\}$ 的一步转移概率矩阵。记 $p_{ij}^{(n)}=P\{X_n=j\mid X_0=i\}$ 为 n 步转移概率，$\boldsymbol{P}^{(n)}=(p_{ij}^{(n)})$ 为 n 步转移概率矩阵。

为了直观了解 Markov 过程中的无后效性，下面用一个青蛙在石头上随机跳动的例子来说明这个问题。用石头来表示状态，X_n则表示青蛙在时刻 n 所处的石头，$(X_n=i)$ 表示青蛙在时刻 n 处在石头 i 上这一随机事件。如果把时刻 n 看做是“现在”，时刻 0，1，…，$n-1$ 表示“过去”，时刻 $n+1$ 表示“将来”，那么式（5.9）表示在过去 $X_0=i_0, \cdots, X_{n-1}=i_{n-1}$及现在 $X_n=i_n$ 的条件下，青蛙在将来时刻 $n+1$ 跳到石头 i_{n+1}的条件概率，只依赖于现在发生的事件 $(X_n=i_n)$，而与过去

历史曾经发生过的事件无关。简而言之，在已知“现在”的条件下，“将来”与“过去”是独立的。$p_{ij}(n)$ 表示青蛙在时刻 n 由石头 i 出发，于时刻 $n+1$ 跳到石头 j 的条件概率，而齐次性 $p_{ij}(n)=p_{ij}$ 表示此转移概率与时刻 n 无关。

2. 以青蛙跳动为例说明状态转移概率

假设青蛙在3块石头上随机跳动（如图5.3所示），且跳动具有无记忆性的特点，也就是无后效性。状态转移概率矩阵 $\boldsymbol{P}$ 为

$$\boldsymbol{P}=\begin{bmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4}\\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4}\end{bmatrix}$$

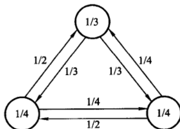


图5.3 青蛙跳动图示

设 $\boldsymbol{\Pi}(t)=[\pi_1(t),\pi_2(t),\cdots,\pi_n(t)]$ 表示 t 时刻青蛙处在各石头上的概率分布向量，$\boldsymbol{\Pi}(t)$ 是一个行向量，则在 $t+1$ 时刻有

$$\boldsymbol{\Pi}(t+1)=\boldsymbol{\Pi}(t)\cdot P \tag{5.12}$$

其中，$\pi_i(t)$ 表示 t 时刻青蛙处在石头 i 上的概率，$n=3$。

若青蛙0时刻处在第1块石头上，也即是，在0时刻，有

$$\boldsymbol{\Pi}(0)=[1\quad 0\quad 0]$$

于是，根据状态转移概率矩阵 P 可计算青蛙在时刻1，2，…处于各石头上的概率向量

$$\boldsymbol{\Pi}(1)=\boldsymbol{\Pi}(0)\cdot P=\begin{bmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\end{bmatrix}$$

$$\boldsymbol{\Pi}(2)=\boldsymbol{\Pi}(1)\cdot P=\cdots$$

假设系统在 $t+1$ 时刻达到稳态，则存在

$$\lim_{t\to\infty}\boldsymbol{\Pi}(t+1)=\lim_{t\to\infty}\boldsymbol{\Pi}(t)=\boldsymbol{\Pi}$$

其中，$\boldsymbol{\Pi}=[\pi_1,\pi_2,\cdots,\pi_n]$ 为系统达到稳态时的状态概率分布向量。由

根据式（5.12）可得

$$\boldsymbol{\Pi}(t+1)=\boldsymbol{\Pi}(t)\cdot T\Rightarrow\boldsymbol{\Pi}=\boldsymbol{\Pi}\cdot T$$

又

$$\sum_{i=1}^{n}\pi_i=1$$

故可得如下线性方程组

$$\begin{cases}[\pi_1,\pi_2,\cdots,\pi_n]\boldsymbol{P}=[\pi_1,\pi_2,\cdots,\pi_n]\\ \pi_1+\pi_2+\cdots+\pi_n=1\end{cases}\tag{5.13}$$

值得注意的是，由于状态转移矩阵 $\boldsymbol{P}$ 的秩 $R(\boldsymbol{P})=n-1$，故方程组（5.13）存在唯一的解。

在上面给出的小例子中

$$\begin{cases}\frac{1}{3}\cdot\pi_1+\frac{1}{2}\cdot\pi_2+\frac{1}{4}\cdot\pi_3=\pi_1\\ \frac{1}{3}\cdot\pi_1+\frac{1}{4}\cdot\pi_2+\frac{1}{2}\cdot\pi_3=\pi_2\\ \frac{1}{3}\cdot\pi_1+\frac{1}{4}\cdot\pi_2+\frac{1}{2}\cdot\pi_3=\pi_3\\ \pi_1+\pi_2+\pi_3=1\end{cases}$$

解这个线性方程组，求得

$$\boldsymbol{\Pi}=\begin{bmatrix}\frac{16}{57}\\ \frac{20}{57}\\ \frac{21}{57}\end{bmatrix}^{\mathrm{T}}$$

从这个计算结果可以看出，当系统达到稳态时，青蛙跳到第三块石头上的机会要多一些，而跳到第一块石头上的概率最小。

3. SA 算法的 Markov 描述

下面考察一下模拟退火算法的搜索过程，算法从一个初始状态开始后，每一步状态转移均是在当前状态 i 的邻域 $N(i)$ 中随机产生一个新状态 j，然后以一定的概率进行接受的。可见，接受概率仅依赖于新状态和当前状态，并由温度加以控制。因此，SA 算法对应一个 Markov 链。若固定每一温度 T_k，算法计算 Markov 链的变化直至平稳分布，然后降低温度，则称这种 SA 算法是齐次的。若无需各温度下算法均达到平稳分布，但温度需按一定的速率下降，则这种 SA 算法是非齐次的或非平稳的。这里仅对齐次 SA 算法的收

敛性进行分析。

Markov 链可以用一个有向图 $G(V, E)$ 表示，其中 V 为所有状态构成的顶点集，$E=\{(i, j) \mid i, j \in V, j \in N(i)\}$ 为边集。

记 a_{ij} 为当前状态 i 接受状态 j 的概率，按照 SA 的计算方法，接受概率如下：

当 $i>j$ 时有 $f(i)>f(j)$，SA 进行无条件转移，$a_{ij}(t)=1$

当 $i<j$ 时有 $f(j)>f(i)$，SA 进行有条件转移，$0<a_{ij}(t)=\exp\left(-\frac{f(j)-f(i)}{t}\right)<1$

其中 $f(\cdot)$ 为目标函数，t 为温度参数。

以上讨论的仅仅是在状态 i 已经选择了状态 j 时接受状态 j 的概率，那么在状态 i 究竟有多大的概率选择状态 j 呢？这是下面要讨论的问题。

记 g_{ij} 为由状态 i 选择状态 j 的概率，则

$$g_{ij}=\begin{cases}\dfrac{g(i, j)}{g(i)}, & j \in N(i) \\ 0, & j \notin N(i)\end{cases}$$

其中

$$g(i)=\sum_{j \in N(i)} g(i, j)$$

它通常与温度无关。若新状态在当前状态的邻域中以等同概率选择，则

$$\frac{g(i, j)}{g(i)}=\frac{1}{|N(i)|}$$

其中，$|N(i)|$ 为状态 i 的邻域中状态总数。

状态 i 到状态 j 的状态转移发生的条件是 i 选择 j 并接受 j，记 p_{ij} 为由状态 i 到状态 j 的转移概率，则根据上面两个概率 g_{ij} 和 a_{ij} 就能够计算转移概率。对于任意 i, j，有

（1）当 $j \neq i$ 且 $j \in N(i)$ 时

$$p_{ij}(t)=a_{ij}(t) \cdot g_{ij}$$

（2）当 $j \neq i$ 且 $j \notin N(i)$ 时

$$p_{ij}(t)=0$$

（3）当 $j=i$ 时

$$p_{ij}(t)=1-\sum_{k \neq i} a_{ik}(t) \cdot g_{ik}$$

因此，状态转移概率矩阵 P 为

$$P = \begin{bmatrix} 1 - \sum_{k \neq 1} a_{1k}g_{1k} & g_{12} & g_{13} & \cdots & g_{1N} \\ g_{21} & 1 - \sum_{k \neq 2} a_{2k}g_{2k} & g_{23} & \cdots & g_{2N} \\ \vdots & \vdots & \vdots & & \vdots \\ g_{N1} & g_{N2} & g_{N3} & \cdots & 1 - \sum_{k \neq N} a_{Nk}g_{Nk} \end{bmatrix} \tag{5.14}$$

若存在这样一个优化问题

$$\min f(x)$$

其中，$x \in S$，S 是有限集，设 $|S| = N$ 表示 S 集中的元素个数为 N，在不失一般性的前提下，让状态按目标函数值进行升序编号，即

$$f(x_1) \leqslant f(x_2) \leqslant \cdots \leqslant f(x_N)$$

下面分两种情况来讨论状态转移概率矩阵 P。

第一种情况：当 $t \to \infty$ 时，

对于 $\forall j > i$，可做如下推导

$$\left.\begin{matrix} j > i \Rightarrow f(j) \geqslant f(i) \\ t \to \infty \end{matrix}\right\} \Rightarrow -\frac{f(j) - f(i)}{t} \to 0 \Rightarrow \mathrm{e}^{-\frac{f(j)-f(i)}{t}} \to 1$$

也就是

$$a_{ij}(t) \to 1$$

对于 $\forall j < i$，由于 $f(j) \leqslant f(i)$，所以进行无条件转移，也就是

$$a_{ij}(t) = 1$$

因此

$$\lim_{T_k \to \infty} a_{ij}(t) = 1, \forall j \neq i$$

根据上面的推导，易知当 $t \to \infty$ 时，从当前状态 i 向状态空间 S 内其他所有的状态进行转移的概率都相等且大于 0，故当前状态 i 的邻域中的状态总数为 $|N(i)| = N - 1$，且

$$g_{ij} = \frac{1}{N-1}, \quad \forall j \neq i$$

根据式（5.14）可知，当 $t \to \infty$ 时

$$p_{ij}(T_k) = \begin{cases} a_{ij}(T_k)g_{ij} = 1 \cdot \dfrac{1}{N-1} = \dfrac{1}{N-1}, j \neq i \\ 1 - \sum\limits_{k \neq i} a_{ik}(T_k)g_{ik} = 1 - (N-1) \cdot \dfrac{1}{N-1} = 0, j = i \end{cases}$$

因此，当 $t \to \infty$ 时，状态转移概率矩阵 P 为

$$
\boldsymbol{P}=\begin{bmatrix} 0 & & \frac{1}{N-1} \\ & 0 & \\ & & \ddots & \\ \frac{1}{N-1} & & 0 \end{bmatrix}
$$

第二种情况：当 $t \to 0$ 时，

对于 $\forall j>i$，可作如下推导

$$
\left.\begin{array}{c} j>i \Rightarrow f(j) \geqslant f(i) \\ t \to 0 \end{array}\right\} \Rightarrow -\frac{f(j)-f(i)}{t} \to -\infty \Rightarrow \mathrm{e}^{-\frac{f(j)-f(i)}{t}} \to 0
$$

也就是

$$
a_{ij}(t) \to 0
$$

对于 $\forall j<i$，由于 $j<i \Rightarrow f(j) \leqslant f(i)$，所以进行无条件转移，也就是

$$
a_{ij}(t)=1
$$

因此

$$
\lim_{t \to 0} a_{ij}(t)=\begin{cases} 0, & j>i \\ 1, & j<i \end{cases}
$$

根据式（5.14）可知，当 $t \to 0$ 时

$$
p_{ij}(t)=\begin{cases} a_{ij}(t) g_{ij}=0 \cdot g_{ij}=0, j>i \\ a_{ij}(t) g_{ij}=1 \cdot g_{ij}=g_{ij}, j<i \\ 1-\sum\limits_{k \neq i} a_{ik}(t) g_{ik}=1-\sum\limits_{k=1}^{i-1} g_{ik}, j=i \end{cases}
$$

因此，当 $t \to 0$ 时，状态转移概率矩阵 $\boldsymbol{P}$ 为

$$
\boldsymbol{P}=\begin{bmatrix} 1 & & & & \\ g_{21} & 1-\sum\limits_{k=1}^{1} g_{2k} & & 0 & \\ g_{31} & g_{32} & 1-\sum\limits_{k=1}^{2} g_{3k} & & \\ \vdots & \vdots & \vdots & \ddots & \\ g_{N1} & g_{N2} & g_{N3} & \cdots & 1-\sum\limits_{k=1}^{N-1} g_{Nk} \end{bmatrix}
$$

此时，$\boldsymbol{P}$ 为一个下三角矩阵，值得注意的是 $\boldsymbol{P}$ 的第一个行向量是 $[1 \quad 0 \quad \cdots \quad 0]$，可见当 $t \to 0$ 时，任何状态一旦到达状态1将无法转出，这种情况称为状态1是“捕捉的”，也就是说当青蛙跳到第1块石头上后就无法跳出了。

无论从实际还是从直观上来看，模拟退火算法要实现全局收敛，它必须满足以下条件：

① 状态可达性，也就是说，无论起点如何，任何状态都是可以到达的。

② 初值鲁棒性，由于初值的选择具有非常大的随机性，因此，算法达到全局最优应不依赖初值。

③ 极限分布的存在性，包含两个方面的内容：其一是当温度不变时，其 Markov 链的极限分布存在；其二是当温度趋近于 0 时，其 Markov链也有极限分布，且最优状态的极限分布和为 1。

下一节就从理论上来分析 SA 算法的收敛性。

5.4.2 SA 的收敛性分析

引理 5.1：当 $T_k \to 0$ 时，系统达到稳态时的状态概率分布向量 $[1 \quad 0 \quad \cdots \quad 0]$。

证明：

设 $\boldsymbol{\Pi} = [\pi_1 \quad \pi_2 \quad \cdots \quad \pi_N]$ 为系统达到稳态时的状态概率分布向量，其中 π_i 是稳态时系统处于状态 i 的概率，$\pi_i \geqslant 0$，$i = 1, 2, \cdots, N$。

因为系统达到稳态，所以有

$$\boldsymbol{\Pi} = \boldsymbol{\Pi} \cdot P$$

当 $T_k \to 0$ 时

$$P = \begin{bmatrix} 1 & & & & \\ g_{21} & 1 - \sum_{k=1}^{1} g_{2k} & & 0 & \\ g_{31} & g_{32} & 1 - \sum_{k=1}^{2} g_{3k} & & \\ \vdots & \vdots & \vdots & \ddots & \\ g_{N1} & g_{N2} & g_{N3} & \cdots & 1 - \sum_{k=1}^{N-1} g_{Nk} \end{bmatrix}$$

因此

$$[\pi_1 \quad \pi_2 \quad \cdots \quad \pi_N] = [\pi_1 \quad \pi_2 \quad \cdots \quad \pi_N] \cdot \begin{bmatrix} 1 & & & & \\ g_{21} & 1-\sum_{k=1}^{1} g_{2k} & & 0 & \\ g_{31} & g_{32} & 1-\sum_{k=1}^{2} g_{3k} & & \\ \vdots & \vdots & \vdots & \ddots & \\ g_{N1} & g_{N2} & g_{N3} & \cdots & 1-\sum_{k=1}^{N-1} g_{Nk} \end{bmatrix}$$

$$\Rightarrow \pi_1 = \pi_1 + \pi_2 \cdot g_{21} + \cdots + \pi_N \cdot g_{N1}$$

$$\Rightarrow \pi_1 = \pi_1 + \sum_{i=2}^{N} \pi_i \cdot g_{i1}$$

$$\Rightarrow \sum_{i=2}^{N} \pi_i \cdot g_{i1} = 0$$

可见，当 $i>1$ 时，$\pi_i \geqslant 0$，$g_{i1} \geqslant 0 \Rightarrow$ 若 $g_{i1}>0$，则 $\pi_i = 0$。

因此，当 $T_k \to 0$ 时，系统达到稳态时的状态概率分布向量 $\boldsymbol{\Pi} = [1 \quad 0 \quad \cdots \quad 0]$。

证毕。

定理 5.1：若选择概率矩阵对称，即对于 $\forall i \neq j$，存在 $g_{ij} = g_{ji}$，则当达到热平衡时，对所有 $T_k > 0$ 存在

$$\boldsymbol{\Pi}(T_k) = \pi_1(T_k)[1 \quad a_{12}(T_k) \quad a_{13}(T_k) \quad \cdots \quad a_{1N}(T_k)]$$

证明：

当在温度 T_k 下达到热平衡时，有

$$\pi_i(T_k)p_{ij}(T_k) = \pi_j(T_k)p_{ji}(T_k)$$

当 $i=1$ 时

$$\pi_1(T_k)p_{1j}(T_k) = \pi_j(T_k)p_{j1}(T_k)$$
$$\pi_1(T_k)g_{1j}a_{1j}(T_k) = \pi_j(T_k)g_{j1}a_{j1}(T_k)$$

由于

$$j>1 \Rightarrow f(j) \geqslant f(1) \Rightarrow \forall T_k, \exists a_{j1} = 1$$

又

$$g_{1j} = g_{j1}$$

所以

$$\pi_1(T_k)a_{1j}(T_k) = \pi_j(T_k), \ j = 2, 3, \cdots, N$$

因此，对于所有 $T_k > 0$，当达到热平衡时

$$\boldsymbol{\Pi}(T_k)=[\pi_1(T_k)\quad \pi_2(T_k)\quad \pi_3(T_k)\quad \cdots\quad \pi_N(T_k)]$$
$$=[\pi_1(T_k)\quad \pi_1(T_k)a_{12}(T_k)\quad \pi_1(T_k)a_{13}(T_k)\cdots\pi_1(T_k)a_{1N}(T_k)]$$
$$=\pi_1(T_k)[1\quad a_{12}(T_k)\quad a_{13}(T_k)\quad \cdots\quad a_{1N}(T_k)]$$

证毕。

由以上定理可知，当 T_k 趋近于 0 时，对于所有状态 $i>1$，有 $a_{1i}(T_k)$ 趋近于 0，$\pi_1(T_k)$ 趋近于 1，即 SA 算法对应的 Markov 过程将以概率 1 收敛于状态 1，即目标值小的状态。

5.5 应用案例

5.5.1 成组技术中加工中心的组成问题

1. 问题描述

成组技术中加工中心的组成问题：设有 m 台机器，要组成若干个组成中心，每个加工中心可最多有 q 台机器、最少 p 台机器，有 n 种工件要在这些机器上加工，已知工件和机器的关系矩阵 A，

$$A=[a_{ij}]_{m\times n},\quad a_{ij}=\begin{cases}1, & \text{机器 } i \text{ 为工件 } j \text{ 所需}\\ 0 & \text{其他}\end{cases}$$

问如何组织加工中心，才能使总的各中心的机器相似性最好？

用 k 表示可能的加工中心数，则存在

$$k_{\min}\leqslant k\leqslant k_{\max}$$

其中，$k_{\min}=\left[\dfrac{m}{q}\right]$；$k_{\max}=\left[\dfrac{m}{p}\right]$；$[V^+]$ 表示返回一个小于 V^+ 的最大整数。

用 S_{ij}表示机器 i 与机器 j 的相似系数，则

$$S_{ij}\in[0,\ 1],\ \text{且}\ S_{ij}=\begin{cases}\dfrac{n_{ij}}{n_i+n_j-n_{ij}} & ,\ i\neq j\\ 0 & ,\ i=j\end{cases}$$

其中，n_{ij}为工件需在机器 i 和 j 上加工的数量；n_i为工件需在机器 i 上加工的数量。举例来说，假设 8 个工件在机器 i 和 j 上加工，工件和机器的关系矩阵 A 为

$$A=\begin{bmatrix}1&1&1&0&0&0&1&1\\0&0&1&1&1&0&1&0\end{bmatrix}\begin{matrix}i & n_i=5\\ j & n_j=3\end{matrix},\quad n_{ij}=2$$

于是

$$S_{ij}=\frac{2}{5+4-2}=\frac{2}{7}$$

2. 模型建立

决策变量有两个：x_{ik}用来表示机器 i 是否指定于加工中心 k，y_k表示是否组成加工中心 k，即

$$x_{ik}=\begin{cases}1, & \text{机器 } i \text{ 指定于中心 } k \quad i=1,2,\cdots,m\\ 0, & \text{其他} \quad k=1,2,\cdots,k_{\max}\end{cases}$$

$$y_k=\begin{cases}1, & \text{组成中心 } k\\ 0, & \text{不组成中心 } k\end{cases} \quad k=1,2,\cdots,k_{\max}$$

根据决策变量，建立加工中心成组优化的数学模型如下：

$$\max\sum_{k=1}^{k_{\max}}\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}S_{ij}x_{ik}x_{jk} \tag{5.15}$$

$$\text{s. t.}\ \sum_{k=1}^{k_{\max}}x_{ik}=1, i=1,2,\cdots,m \tag{5.16}$$

$$\sum_{i=1}^{m}x_{ik}\leqslant qy_k, k=1,2,\cdots,k_{\max} \tag{5.17}$$

$$\sum_{i=1}^{m}x_{ik}\geqslant py_k, k=1,2,\cdots,k_{\max} \tag{5.18}$$

$$x_{ik},y_k=0\text{ 或 }1,\forall i,k \tag{5.19}$$

目标函数是使成组的相似性极大化，也就是期望将所有相似的机器放在同一个中心；约束条件（5.16）用于指定唯一性，以保证每个机器只能放在一个加工中心；约束条件（5.17）保证每个中心的机器数要小于其最大容量 p 台；约束条件（5.18）保证每个中心的机器数要大于其最大容量 q 台；式（5.19）为决策变量。

这是一个典型的二次指派问题，其中决策变量有 $m\times k_{\max}+k_{\max}$个，约束条件有 $m+2k_{\max}+m\times k_{\max}+k_{\max}$个，用普通的二次 **0-1** 规划方法求解，由于变量数较多，处理起来比较困难，因此采用模拟退火方法对这个模型进行求解。

3. 模拟退火算法

状态表达：采用自然数编码作为状态表达方法，设 $x_i=k$ 表示机器 i 在中心 k，则 $x=[x_1, x_2, \cdots, x_m]$ 就可以表示一个状态，利用这种编码方法就使得原问题等价于一个 K 分图问题。

目标函数：由于该问题是一个有约束的优化问题，而 SA 用于求解无约束问题，故首先需要将上述模型转化为如下无约束模型：

$$\max z=\sum_{v_k\in P_k}\sum_{i\in V_k}\sum_{j\in V_k}s_{ij}-\left(\frac{\alpha}{T_k}\right)\sum_{V_i\in P_k''}(p-|v_i|)^2-\left(\frac{\beta}{T_k}\right)\sum_{V_j\in P_k'}(|v_j|-q)^2$$

其中，α 和 β 是罚因子；T_k 是温度参数，可见随着算法的运行（T_k 逐渐下降），罚因子会逐渐增大，这就保证了算法在开始阶段进行广域搜索，到了终止阶段进行局域搜索。

$P_k=\{v_1, v_2, \cdots, v_k\}$表示集类，它是集合的集合；$v_k=\{i|v_i=k\}$集合；$P'_k=\{v_i\in P_k||v_k|>q\}$机器数超标的中心集合；$P''_k=\{v_i\in P_k||v_k|<p\}$ 机器数不够的中心集合。

举例来说，对于这样一个问题：

$$p=2, q=5, \boldsymbol{x}=[2\ \ 1\ \ 1\ \ 2\ \ 3\ \ 3\ \ 2\ \ 2\ \ 1\ \ 3]$$

此时，

$$\boldsymbol{v}_1=\{2, 3, 9\}, |\boldsymbol{v}_1|=3$$
$$\boldsymbol{v}_2=\{1, 4, 7, 8\}, |\boldsymbol{v}_2|=4$$
$$\boldsymbol{v}_3=\{5, 6, 10\}, |\boldsymbol{v}_3|=3$$

邻域：从当前状态 $\boldsymbol{x}=[x_1, x_2, \cdots, x_m]$ 中随机选择一个 x_i，改变其位值，这样就产生了一个邻域点，故邻域的大小为 $m(k_{\max}-1)$。

内循环次数：$n(T_k)=m(k-1)$，其中 k 为迭代指标。

降温函数：$T_{k+1}=\dfrac{T_k}{1+\alpha T_k}$，其中 $\alpha=\dfrac{\ln(1+\delta)}{3\delta_{f(x)}}$，$\delta_{f(x)}=\sqrt{\sum_{i=1}^{n(T_k)}(f_i-\bar{f})^2}$，$\delta$ 为一个控制参数。

可见，$\delta_{f(x)}\uparrow\Rightarrow\alpha\downarrow\Rightarrow$，下降慢，而当 T_k很大时，温度下降的速度很快。

5.5.2 准时化生产计划问题

1. 准时化生产计划

20 世纪 70 年代末，准时化（JIT）生产技术在日本成功地提出。由于它能大幅度地减少生产和存储费用，因此吸引了许多学者去研究 JIT 环境下提前/拖期生产计划问题。半无限规划模型准确地描述了提前/拖期生产计划问题，只是模型中有无限多非凸约束，求解比较困难。目前求解该模型比较有效的方法是遗传算法，但是用遗传算法求解半无限规划模型存在一些缺点，如占用内存大、计算量大、运算时间长、不能保证解的最优性等。下面用模拟退火结合启发式算法和最速下降法求解半无限规划模型，此算法可以跳出局部最优解，并且以概率 1 收敛于全局最优解。

2. 半无限规划模型的建立

假设制造商在［0, T］计划生产时期内收到 n 个订货。在 JIT 环境

下，决策者希望计划生产完成时间尽可能接近顾客要求的交货期。通过使用一个二次惩罚函数，可以得出提前/拖期生产的半无限规划模型

$$\min_x \sum_{i=1}^{n} \alpha_i (x_i - d_i)^2 \tag{5.20}$$

$$\text{s. t.} \sum_{i=1}^{n} R_i(t,x_i) \leqslant G(t), t \in [0,T] \tag{5.21}$$

$$0 \leqslant L_i \leqslant x_i \leqslant T, i = 1,2,\cdots,n \tag{5.22}$$

其中，L_i是订货 i 的生产周期，顾客要求的交货期是 d_i，计划完成时间是 x_i，资源需求函数由 $R_i(t,\ x_i)$，$t\in\ [0,\ T]$ 表示。T 时刻可采用的资源为 $G(t)$；α_i 是惩罚系数。根据生产实际，假设资源需求函数是近似于正态分布的曲线，即

$$R_i(t,\ x_i) = a_i \exp\left[-\frac{(t - x_i + b_i)^2}{c_i} \right],\ i=1,\ 2,\ \cdots,\ n \tag{5.23}$$

其曲线如图 5.4 所示。等式（5.23）只是实际资源需求函数的一个近似表达式。其中 $a_i = P_i\left(\frac{\sqrt{2\pi}L_i}{4}\right)$，$b_i = \frac{L_i}{2}$，$c_i = \frac{L_i^2}{8}$。

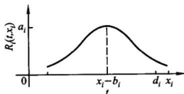


图 5.4 资源需求函数曲线图

设开始可采用的资源是 g_0，它随时间的增加以速率 β 增长。假设在 $[0,\ T]$ 内有 m 个遗留工作，其中每一个工作需要一定的资源，由下面的近似等式确定（$j=1,\ 2,\ \cdots,\ m$）。

$$Q_j(t) = q_j \exp\left[-\frac{(t - g_j)^2}{h_j} \right],\ t\in\ [0,\ T]$$

其中参数 q_j，g_j，h_j与 a_i，b_i，c_i同样的方法求得。因此，可采用的资源函数变为

$$G(t) = g_0 \exp(\beta t) - \sum_{j=1}^{m} Q_j(t), t \in [0,T]$$

3. 模拟退火算法

这里采用模拟退火算法求解半无限规划模型，分 3 个阶段：先用启发式算法得出一个较好的初始解，再用模拟退火算法结合最速下降法对初始解中时间 x_i的顺序进行优化，最后用模拟退火算法对第二阶段的结

果进行微调，求最优解。由于模拟退火算法适用于无约束函数，这里采用一种非精确方法将约束通过惩罚项结合到目标函数中。

由于 $R_i(t, x_i)$ $(i=1, 2, \cdots, n)$ 是近似于正态分布的曲线，所以可以证明不等式（5.21）左侧最多有 n 个局部极大值，并且位于 $[x_i - L_i, x_i]$ $(i=1, 2, \cdots, n)$ 内。设

$$t_j = \arg\max_t \left\{ \sum_{i=1}^{n} R_i(t, x_i) - G(t) \mid x_j - L_j < t < x_j \right\}$$

$$\Phi(t_j, X) = \sum_{i=1}^{n} R_i(t_j, x_i) - G(t_j), j = 1, 2, \cdots, n$$

那么 $\Phi(t_j, X) > 0$ 就意味着违反（SIP）的约束（5.21），对约束（5.21）若给定一个允许误差 ε，则所有违反约束（5.21）的点的集合为 $V = \{t_j \mid j=1, 2, \cdots, n, \Phi(t_j, X) > \varepsilon\}$。那么，将约束结合到目标函数中得到新的目标函数为

$$\min_x \sum_{i=1}^{n} \alpha_i (x_i - d_i)^2 + \mu \sum_{t_j \in V} \Phi(t_j, X)$$

其中 μ 是足够大的惩罚系数。

（1）用启发式算法求初始解

如果随机产生一个初始解，高温退火过程比较长。为减少运算时间，用较快的启发式来代替高温退火，得出一个比较好的初始解。根据实际经验，为减少总的惩罚，对于惩罚大的、资源需求少的订货，应该尽量减少提前/拖期的时间，即以 $\frac{\alpha_i}{P_i}$ $(i=1, 2, \cdots, n)$ 的值确定优先级，$\frac{\alpha_i}{P_i}$ 大的先考虑。具体算法如下。

第 1 步：确定计划水平 T

$$T = \max\left\{ \frac{\eta\left(\sum_{i=1}^{n} P_i + \sum_{j=1}^{m} q_j\right)}{g0}, \max\left[d_i \mid i=1, 2, \cdots, n\right] \right\}$$

第 2 步：计算 $u_i = \frac{\alpha_i}{P_i}$ $(i=1, 2, \cdots, n)$，并对 u_i 从大到小排序得 $u_1, u_2, \cdots, u_n$

第 3 步：以 $u_1, u_2, \cdots, u_n$ 为优先级分别求各订货的生产完成时间 x_j。

① $x_j = d_j$，如果 $\Phi(t_j, X) > 0$ $(t_j \in [d_j - L_j, d_j])$，转②；

② 对 $k=1, 2, \cdots, d_j - L_j$，依次计算 $y_j = d_j - k$ 直到 $\Phi(t_j, X) < 0$。

计算提前惩罚 $a=\alpha_j(d_j-y_j)$，转③；否则 $a=M$（M 是大的数）；

③ 对 $k=1, 2, \cdots, T-d_j$，依次求 $z_j=d_j+k$ 直到 $\Phi(t_j, Z)<0$，计算拖期惩罚 $b=\alpha_j(z_j-d_j)$；否则 $b=M$；

④ 比较 a、b，取最小惩罚的 x_j。如果 $a \leqslant b$，则 $x_j=y_j$；否则 $x_j=z_j$。

（2）用模拟退火算法（SA1）调优 x_i（$i=1, 2, \cdots, n$）顺序

由于模拟退火算法接受使目标值变坏的解，所以最终解有可能比计算过程中遇到的最好解坏。对模拟退火算法稍加改进，即保存遇到的最好可行解。对模拟退火算法中涉及的降温规则和热平衡条件做如下规定。

① 降温规则：通过对 T_k 乘以一个接近 1 的数 r 来减少 T_k。

② 热平衡条件：在每种温度下，转换次数应该等于邻域的大小，本算法中，任意 x_i 与 x_j 交换一次作为邻域中的一个解，故至少应交换 C_n^2 次。

算法 SA1

第 1 步：由启发式算法得初始解 $X_0=(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$，计算点 X_0 目标函数 $f(X_0)$，并将 X_0 作为最好解保存。

第 2 步：设定初始温度 T_0，终止温度 T_f，令 $k=0$，$T_k=T_0$。

第 3 步：产生随机解（任意交换 x_i，x_j）X_1，$k=k+1$，计算 $\Delta f=f(0)-f(1)$。

第 4 步：如果 $\Delta f<0$，令 $X_0=X_1$，$f(X_0)=f(X_1)$，必要时更新最好解。转第 6 步。

第 5 步：产生 $\xi \in U(0, 1)$，如果 $\exp\left(-\dfrac{\Delta f}{T_k}\right)>\xi$，则 $X_0=X_1$，$f(X_0)=f(X_1)$。

第 6 步：如果 $k=C_n^2$，则令 $k=0$ 转第 7 步，否则转第 3 步。

第 7 步：令 $T_k=rT_k$，如果 $T_k \leqslant T_f$，停止；否则转第 3 步。

第 3 步交换 x_i 与 x_j 后，即使是最佳顺序，目标函数值也可能产生惩罚，所以每得出一个顺序，如果 $\Delta f>0$，先用最速下降法求该顺序下的近优解，得出的结果作为 X_1，继续后面的步骤。

（3）用模拟退火（SA2）求最优解

首先需要将连续问题离散化。因为模拟退火方法适用于解决离散问题，即自变量取离散的点。而半无限规划是连续型问题，其自变量 $x_i \in [0, T]$（$i=1, 2, \cdots, n$），所以，在 x_i 的邻域 $(x_i-\delta, x_i+\delta)(\delta \leqslant 1)$ 内取离散的点，从而将连续问题离散化。即 x_i 的邻域为

$$N(x_i)=\{y_i \mid y_i=x_i-\delta+2\delta\gamma,$$
$$\gamma=0, 0.1, 0.2, \cdots, 0.9\},\ i=1, 2, \cdots, n$$

由此得第 3 阶段模拟退火算法如下：

算法 SA2

第 1 步：以 SA1 计算结果作为初始解 X_0，并作为最好解保存，计算 $f(X_0)$。

第 2 步：设定 T_0，T_f，令 $k=0$，$T_k=T_0$。

第 3 步：在 X_0 邻域内产生随机解 $Y=(y_1, y_2, \cdots, y_n)^T$，计算 $\Delta f=f(Y)-f(X_0)$，$k=k+1$。

第 4 步：与算法 SA1 的第 4 步至第 7 步相同。

4. 计算举例

考虑一个建筑公司收到 10 个建造楼房的订货，主要资源约束是人力。现有人力水平是 100 千时/周，并预计以 $\beta=0.005$ 的速率增加。有两个以前计划中遗留的工作 A 和 B，对 A，人力需求是 200 kh，它必须在第 40 周完成，需要 20 周的加工时间；对 B，人力需求 300 kh，它必须在第 70 周完成，需要 40 周加工时间。建筑周期，人力需求，合同价格（即惩罚系数）及对每个订货顾客要求的交货期见表 5.2。分别利用上面介绍的模拟退火算法和遗传算法进行计算，两种方法的运算结果在表 5.2 的后两列。用遗传算法计算的目标函数值是 114 779，用模拟退火方法计算的目标函数值是 112 797.6。对于 6 个订货的例子，其订货和计算结果见表 5.3，用遗传算法求解的目标值是 15 154，用模拟退火方法求解的目标值是 14 355.9。

表 5.2　10 个订货的数据和计算结果

订货	生产周期	资源需求	合同价格	要求交货期	生产完成时间	
i	L_i	P_i	α_i	d_i	x_i (GA)	x_i (SA)
1	20	400	10	25	30.74	32.63
2	20	900	18	30	20.76	20.43
3	30	800	12	35	41.88	38.08
4	40	800	15	40	40.79	42.03
5	25	1 000	28	40	52.50	54.68
6	20	1 200	20	40	86.77	86.06
7	50	2 000	30	50	84.38	83.35
8	10	300	18	15	10.00	10.00
9	20	400	9	50	75.75	42.67
10	60	1 500	30	60	87.28	90.34

表 5.3 6 个订货的数据和计算结果

订货 i	生产周期 L_i	资源需求 P_i	合同价格 α_i	要求交货期 d_i	生产完成时间 x_i (GA)	x_i (SA)
1	20	400	10	25	29.8	28.7
2	20	900	18	30	20.0	20.0
3	30	800	12	35	46.0	39.9
4	40	800	15	40	40.0	52.7
5	25	1 000	28	40	56.3	57.2
6	60	1 500	30	60	71.9	66.9

问题与思考

1. 旅行商问题可简述如下：找一条经过 n 个城市的巡回（每个城市过且只过一次），极小化总的路程。其中，d_{ij}为城市 i 与 j 间的距离。试按模拟退火算法设计一个求解该问题的算法（包括状态表达，邻域定义，算法步骤），并画出算法框图。

2. 工作指派问题可简述如下：n 个工作可以由 n 个工人分别完成。工人 i 完成工作 j 的时间为 d_{ij}。问如何安排可使总的工作时间达到极小。试按模拟退火算法设计一个求解该问题的算法（包括状态表达，邻域定义，算法步骤），并画出程序框图。

参考文献

[1] Ackley D H, Hinton G E, Seinowski T J. A learning algorithm for Boltzmann machines[J]. Cognitive Science, 1985, 9: 147 - 169.

[2] Chen W H, Srivastava B. Simulated annealing procedures for forming machine cells in group technology[J]. European Journal of Operational Research, 1994, 75: 100 - 111.

[3] Fang, S C, Wu S Y. An inexact approach to solving linear semi-infinite programming problems[J]. Optimization, 1994, 28(8): 291 - 299.

[4] Kirkpatrick S, Gelatt Jr C D, Vecchi M P. Optimization by simulated annealing [J]. Science, 1983, 220: 671 - 680.

[5] Metroplis N, Rosenbluth A, Rosenbluth M et al. Equation of state calculations by fast computing machines[J]. Journal of Cherimal Physics, 1953, 21: 1087 - 1092.
[6] Lundy M, Mess A. Convergence of an annealing. algorithm [J]. Mathematical Programming, 1986, 34: 111 - 124.
[7] 李颖娟，汪定伟. 准时化生产计划的半无限规划模型与模拟退火方法 [J]. 控制与决策，1998，13(5)：603 - 607.
[8] 邢文训，谢金星. 现代优化计算方法（第2版）[M]. 北京：清华大学出版社，2005.

第6章 蚁群算法

蚁群算法是20世纪90年代发展起来一种模仿蚂蚁群体行为的新的智能化算法。该算法引入正反馈并行机制，具有较强的鲁棒性、优良的分布式计算机制、易于与其他方法结合等优点。目前蚁群算法已经渗透到多个应用领域，从一维静态优化问题到多维动态优化问题，从离散问题到连续问题。蚁群算法都展现出优异的性能和广阔的发展前景，成为国内外学者竞相关注的研究热点和课题。本章将系统地介绍蚁群算法的理论、方法和应用，基本的蚁群算法，包括分析基本蚁群算法的机制和原理，介绍基本蚁群算法的数学模型、实现方法；介绍改进的蚁群算法，概括性介绍蚁群算法收敛性研究的成果；在分析蚁群算法与其他仿生优化算法异同的基础上，介绍蚁群算法与遗传算法的融合；最后是蚁群算法的典型应用，总结蚁群算法在各个领域的应用，重点介绍蚁群算法在车辆路径问题和车间调度作业问题中的应用。

6.1 导　言

人类在自然界获得启示，发明了许多试图通过模拟自然生态系统机制来求解复杂优化问题的仿生优化方法，如本书中提到的遗传算法、蚁群算法、粒子群算法、捕食搜索算法等。这些与经典的数学规划截然不同的仿生优化算法的相继出现，大大丰富了优化技术，使许多在人类看来高度复杂的优化问题得到更好的解决。

研究群居性昆虫行为的科学家发现，昆虫在群落一级上的合作基本上是自组织的，在许多场合中尽管这些合作很简单，但它们却可以解决许多复杂的问题。每只蚂蚁的智能并不高，看起来没有集中的指挥，但它们却能协同工作寻找食物。据此，意大利学者 Dorigo M 等人提出一种模拟昆虫王国中蚂蚁群体觅食行为方式的仿生优化算法——蚁群算法（Ant Colony Algorithm，ACA）。该算法引入正反馈并行机制，具有较强的鲁棒性、优良的分布式计算机制、易于与其他方法结合等优点。目前蚁群算法已经渗透到各个应用领域，从一维静态优化问题到多维动态优化问题，从离散问题到连续问题。蚁群算法解决了许多复杂优化和经典NP-C问题，展现出优异的性能和广阔的发展前景。成为国内外学者竞

相关注的研究热点和前沿性课题。

6.1.1 蚁群觅食的特性

那么在自然界中蚂蚁是如何觅食的呢？为什么蚁群总能找到一条从蚁巢到食物源的最短路径？原来蚂蚁会分泌一种叫做信息素（Pheromone）的化学物质，蚂蚁的许多行为受信息素的调控，蚂蚁在运动过程中，能够在其经过的路径上留下信息素，而且能感知这种物质的存在及其浓度，以此指导自己的运动方向。蚂蚁倾向于朝着信息素浓度高的方向移动。

举例说明，蚁巢在 A 点，蚁群发现食物源在 D 点，它们总是会选择最短的直线路径 AD 来搬运食物，如图 6.1(a)所示。如果搬运路线上突然出现障碍物，不管路径长短，蚂蚁按相同的概率选择在图中 B 点 C 点绕过障碍物，如图 6.1(b)所示。由于路径 ABD 的长度小于路径 ACD 的长度，单位时间内通过路径 ABD 的蚂蚁数量大于通过路径 ACD 的蚂蚁数量，则在路径 ABD 上面遗留的信息素浓度比较高，因为蚂蚁倾向于朝着信息素浓度高的方向移动，所以选择路径 ABD 的蚂蚁随之增多，如图 6.1(c)所示。于是，蚁群的集体行为表现出一种信息正反馈现象，即最短路径上走过的蚂蚁越多，则后来的蚂蚁选择该路径的概率就越大，蚂蚁个体之间就是通过这种信息的交流达到寻找食物和蚁穴之间最短路径的目的，如图 6.1(d)所示。

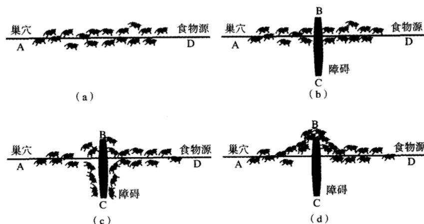


图 6.1 蚁群寻找食物过程

6.1.2 人工蚂蚁与真实蚂蚁的异同

蚁群算法是利用蚁群觅食的群体智能解决复杂优化问题的典型例子。为了使蚁群算法有令人满意的性能，要在真实的蚁群基础上继承些什么，摒弃些什么？下面看一看人工蚂蚁与真实蚂蚁的异同。

1. 相同点

（1）两个群体中都存在个体相互交流的通信机制。真实蚂蚁在经过的路径上留下信息素，用以影响蚁群中的其他个体。且信息素随着时间推移逐渐挥发，减小历史遗留信息对蚁群的影响。同样，人工蚂蚁改变其所经过路径上存储的数字化信息素，该信息素记录了人工蚂蚁当前解和历史解的性能状态，而且可被后继人工蚂蚁读写。数字化的信息素同样具有挥发特性，它像真实的信息量挥发一样使人工蚂蚁逐渐忘却历史遗留信息，在选择路径时不局限于以前人工蚂蚁所存留的经验。

（2）都要完成寻找最短路径的任务。真实蚂蚁要寻找一条从巢穴到食物源的最短路径。人工蚂蚁要寻找一条从源节点到目的节点间的最短路径。两种蚂蚁都只能在相邻节点间一步步移动，直至遍历完所有节点。

（3）都采用根据当前信息进行路径选择的随机选择策略。真实蚂蚁和人工蚂蚁从某一节点到下一节点的移动都是利用概率选择策略实现的。这里概率选择策略是基于当前信息来预测未来情况的一种方法。

2. 不同点

（1）人工蚂蚁具有记忆能力，而真实蚂蚁没有。人工蚂蚁可以记住曾经走过的路径或访问过的节点，可提高算法的效率。

（2）人工蚂蚁选择路径的时候并不是完全盲目的，受到问题空间特征的启发，按一定算法规律有意识地寻找最短路径（如在旅行商问题中，可以预先知道下一个目标的距离）。

（3）人工蚂蚁生活在离散时间的环境中，即问题的求解规划空间是离散的，而真实蚂蚁生活在连续时间的环境中。

6.1.3 蚁群算法的研究进展

1991年，意大利学者Dorigo M等人在法国巴黎召开的第一届欧洲人工生命会议（European Conference on Artificial Life，ECAL）上首次提出了蚁群算法，之后的5年中并没有受到国际学术界的广泛关注。1996年，Dorigo M等人在《IEEE Transactions on Systems，Man，and Cybernetics-Part B》上发表了“Ant system：optimization by a colony of coopera-

ting agents”一文，奠定了蚁群算法的基础。在这之后的5年里，蚁群算法逐渐引起了国际学术界的广泛关注。1998年，Dorigo M 组织在比利时布鲁塞尔召开了第一届蚁群算法国际研讨会（ANTS’98），随后每隔两年都要在布鲁塞尔召开一次蚁群算法国际研讨会。2000年，Gutjahr W J 发表了题为“A graph-based ant system and its convergence”的学术论文，对蚁群算法的收敛性进行了证明。同年，Dorigo M 和 Bonabeau E 等在国际顶级学术杂志《Nature》上发表了蚁群算法研究综述，将这一研究推向国际学术界的前沿。

最近几年，国际顶级学术杂志《Nature》曾多次对蚁群算法的研究成果进行报道，《Future Generation Computer Systems》（Vol. 16，No. 8）和《IEEE Transactions on Evolutionary Computation》（Vol. 6，No. 4）分别于2000年和2002年出版了蚁群算法特刊。

我国对蚁群算法的研究起步较晚，从公开发表论文的时间来看，国内最先研究蚁群算法的是东北大学的张纪会和徐心和。尔后，高尚、汪镭、李艳君、段海滨、陈崚、张勇德和杨勇等都有不俗的工作。段海滨的著作《蚁群算法原理及其应用》则为国内第一本系统介绍研究蚁群算法的学术著作，参见参考文献[20]。目前，蚁群算法的研究已经由单一的TSP领域渗透到多个应用领域。从一维静态优化问题到多维动态优化问题，从离散问题到连续问题。同时蚁群算法在模型改进及与其他仿生优化算法的融合方面也取得了相当丰富的研究成果，展现出广阔的发展前景。

6.2 基本蚁群算法

这一节首先从深层上对基本蚁群算法的机理进行研究，从TSP的角度对基本蚁群算法的数学模型进行分析，并给出具体实现步骤和程序结构框架。然后在引入复杂度概念的基础上，对基本蚁群算法进行复杂度分析。最后讨论参数选择对蚁群算法性能的影响。这节内容是蚁群算法的理论分析部分，也是深入理解蚁群算法、改进蚁群算法、应用蚁群算法的基础。

6.2.1 基本蚁群算法的原理

基本蚁群算法（Ant System，AS）是采用人工蚂蚁的行走路线来表示待求解问题可行解的一种方法。每只人工蚂蚁在解空间中独立地搜索可行解，当它们碰到一个还没有走过的路口时，就随机挑选一条路径前

行，同时释放出与路径长度有关的信息素。路径越短信息素的浓度就越大。当后继的人工蚂蚁再次碰到这个路口的时候，以相对较大的概率选择信息素较多的路径，并在“行走路线”上留下更多的信息素，影响后来的蚂蚁，形成正反馈机制。随着算法的推进，代表最优解路线上的信息素逐渐增多，选择它的蚂蚁也逐渐增多，其他路径上的信息素却会随着时间的流逝而逐渐消减，最终整个蚁群在正反馈的作用下集中到代表最优解的路线上，也就找到了最优解。在整个寻优过程中，单只蚂蚁的选择能力有限，但蚁群具有高度的自组织性，通过信息素交换路径信息，形成集体自催化行为，找到最优路径。图6.2是一个基于蚁群算法的人工蚁群系统寻找最短路径的例子。

如图6.2(a)所示，路径BF，CF和BEC的路程长度d为1，E是路径BEC的中点。假设在每个单位时间内有30只蚂蚁从A来到B，30只蚂蚁从D来到C，每只蚂蚁单位时间内行进路程为1，蚂蚁在行进过程中在单位时间内留下1个浓度单位的信息素，在一个时间段（t，$t+1$）结束后瞬间完全挥发。

如图6.2(b)所示，$t=0$时，在B和C点各有30只蚂蚁，由于此前路径上没有信息素，它们随机地选择路径，在BF、BE、CF和CE上各有15只蚂蚁。

如图6.2(c)所示，$t=1$时，又有30只蚂蚁到达B。它们发现在BF上的信息素浓度为15，BE上信息素浓度为30（是由15只BE走向和15只EB走向的蚂蚁共同留下的），因此选择BE路径的蚂蚁数的期望值是选择BF蚂蚁数的2倍。所以，20只蚂蚁选择BE，10只蚂蚁选择BF。同样的情况发生在C点。这个过程一直持续下去，直到所有人工蚂蚁最终选择最短路径BEC（或CEB）。

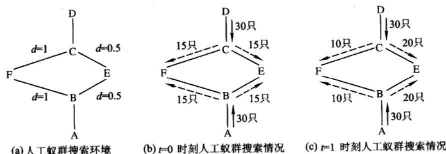


图6.2　人工蚁群路径搜索实例

6.2.2 基本蚁群算法的数学模型

很多文献对基本蚁群算法的详细介绍都是从旅行商问题（Traveling Salesman Problem，简称 TSP）开始。这是因为蚁群觅食的过程与 TSP 问题的求解非常相似，为了便于读者更好地理解蚁群算法的数学模型和实现过程，以 n 个城市 TSP 问题作为背景介绍基本蚁群算法。TSP 问题属于一种典型的组合优化问题，是组合优化问题中最经典的 NP 难题之一，它在蚁群优化算法的发展过程中起着非常重要的作用。

TSP 问题：给定 n 个城市的集合 $C=\{c_1, c_2, \cdots, c_n\}$ 及城市之间旅行路径的长短 $d_{ij}(1\leqslant i\leqslant n, 1\leqslant j\leqslant n, i\neq j)$。TSP 问题是找到一条只经过每个城市一次且回到起点的、最短路径的回路。设城市 i 和 j 之间的距离为 d_{ij}，表示如式（6.1）所示：

$$d_{ij}=[(x_i-x_j)^2+(y_i-y_j)^2]^{\frac{1}{2}} \tag{6.1}$$

TSP 求解中，假设蚁群算法中的每只蚂蚁是具有下列特征的简单智能体。

① 每次周游，每只蚂蚁在其经过的支路（i, j）上都留下信息素。

② 蚂蚁选择城市的概率与城市之间的距离和当前连接支路上所包含的信息素余量有关。

③ 为了强制蚂蚁进行合法的周游，直到一次周游完成后，才允许蚂蚁游走已访问过的城市（这可由禁忌表来控制）。

蚁群算法中的基本变量和常数有：m，蚁群中蚂蚁的总数；n，TSP 问题中城市的个数；d_{ij}，城市 i 和 j 之间的距离，其中 $i, j\in(1, n)$；$\tau_{ij}(t)$，表示 t 时刻在路径（i, j）连线上残留的信息量。在初始时刻各条路径上信息量相等，并设 $\tau_{ij}(0)=const$（$const$ 为常数）。

蚂蚁 k（$k=1, 2, \cdots, m$）在运动过程中，根据各条路径上的信息量决定其转移方向。$p_{ij}^k(t)$ 表示在 t 时刻蚂蚁 k 由城市 i 转移到城市 j 的状态转移概率，根据各条路径上残留的信息量 $\tau_{ij}(t)$ 及路径的启发信息 η_{ij} 来计算的，如式(6.2)所示。表示蚂蚁在选择路径时会尽量选择离自己距离较近且信息素浓度较大的方向。

$$p_{ij}^k(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}{\sum\limits_{s\subset allowed_k}[\tau_{is}(t)]^{\alpha}\cdot[\eta_{is}(t)]^{\beta}}, & j\in allowed_k\\ 0, & \text{其他}\end{cases} \tag{6.2}$$

式中，$allowed_k=\{C-tabu_k\}$ ——表示在 t 时刻蚂蚁 k 下一步允许选择的城市（即还没有访问的城市）；

$tabu_k(k=1, 2, \cdots, m)$ ——表示禁忌表，记录蚂蚁 k 当前已走过的城市；

α——表示信息启发式因子，反映了蚁群在运动过程中所残留的信息量的相对重要程度；

β——表示期望启发式因子，反应了期望值的相对重要程度；

η_{ij}——表示由城市 i 转移到城市 j 的期望程度，被称为先验知识，这一信息可由要解决的问题给出，并由一定的算法来实现，TSP 问题中一般取值为式（6.3）。

$$\eta_{ij}(t)=\frac{1}{d_{ij}} \tag{6.3}$$

对蚂蚁 k 而言，d_{ij}越小，则 $\eta_{ij}(t)$ 越大，$p_{ij}^k(t)$ 也就越大。

为了避免残留信息素过多而淹没启发信息，在每只蚂蚁走完一步或者完成对所有 n 个城市的遍历后，要对残留信息素进行更新处理。$(t+n)$ 时刻在路径 (i, j) 上信息量可按式（6.4）和式（6.5）所示的规则进行调整。

$$\tau_{ij}(t+n)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij}(t) \tag{6.4}$$

$$\Delta\tau_{ij}(t)=\sum_{k=1}^{m}\Delta\tau_{ij}^k(t) \tag{6.5}$$

式中，ρ——表示信息素挥发系数。模仿人类记忆特点，旧的信息将逐步忘却、削弱。为了防止信息的无限积累，ρ 的取值范围为 $[0, 1)$，用 $1-\rho$ 表示信息的残留系数。

$\Delta\tau_{ij}(t)$ ——表示本次循环中路径 (i, j) 上的信息素增量，初始时刻 $\Delta\tau_{ij}(t)=0$。

$\Delta\tau_{ij}^k(t)$——表示第 k 只蚂蚁在本次循环中留在路径 (i, j) 上的信息量。

根据信息素更新策略的不同，Dorigo M 提出了三种不同的基本蚁群算法模型，分别称之为蚁周模型（Ant-Cycle Model），蚁量模型（Ant-Quantity Model）及蚁密模型（Ant-Density Model），三种模型的差别在于 $\Delta\tau_{ij}^k(t)$ 求法不同，下面比较三种模型的异同。

蚁周模型（Ant-Cycle Model）

$$\Delta\tau_{ij}^k(t)=\begin{cases}\dfrac{Q}{L_k}, & \text{第 } k \text{ 只蚂蚁在本次循环中经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{6.6}$$

蚁量模型（Ant-Quantity Model）

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}\dfrac{Q}{d_{ij}}, & \text{第 } k \text{ 只蚂蚁在 } t \text{ 和 } t+1 \text{ 之间经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{6.7}$$

蚁密模型（Ant-Density Model）

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}Q, & \text{第 } k \text{ 只蚂蚁在 } t \text{ 和 } t+1 \text{ 之间经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{6.8}$$

式中，Q——常量，表示蚂蚁循环一周或一个过程在经过的路径上所释放的信息素总量，它在一定程度上影响算法的收敛速度；

L_k——表示第 k 只蚂蚁在本次循环中所走路径的总长度。

区别：式(6.6)利用整体信息，蚂蚁完成一个循环后才更新所有路径上的信息素；式(6.7)和式(6.8)利用局部信息，蚂蚁每走一步就要更新路径上的信息素；式(6.6)蚁周模型在求解 TSP 问题时效果较好，应用也比较广泛。

6.2.3 基本蚁群算法的具体实现

这里的基本蚁群算法是基于蚁周模型（Ant-Cycle Model）的，实现步骤为

第 1 步：初始化参数。时间 $t=0$，循环次数 $N_c=0$，设置最大循环次数 N_{cmax}，令路径 (i,j) 的初始化信息量 $\tau_{ij}(t)=const$，初始时刻 $\Delta\tau_{ij}(0)=0$。

第 2 步：将 m 只蚂蚁随机放在 n 个城市上。

第 3 步：循环次数 $N_c \leftarrow N_c+1$。

第 4 步：令蚂蚁禁忌表索引号 $k=1$。

第 5 步：$k=k+1$。

第 6 步：根据状态转移概率公式(6.2)计算蚂蚁选择城市 j 的概率，$j\in\{C-tabu_k\}$。

第 7 步：选择具有最大状态转移概率的城市，将蚂蚁移动到该城市，并把该城市记入禁忌表中。

第 8 步：若没有访问完集合 C 中的所有城市，即 $k<m$，跳转至第 5 步；否则，转第 9 步。

第 9 步：根据式(6.4)和式(6.5)更新每条路径上的信息量。

第 10 步：若满足结束条件，循环结束输出计算结果；否则清空禁忌表并跳转到第 3 步。

基本蚁群算法的算法框图如图 6.3 所示。

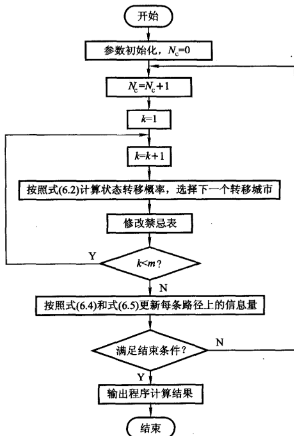


图 6.3 基本蚁群算法的算法框图

6.2.4 基本蚁群算法的复杂度分析

每个组合优化问题都可以通过枚举的方法求得最优解，枚举是以时间为代价的，即使是在计算机软硬件技术高速发展的今天，满足大规模问题求解所要求的计算时间和存储空间仍然是非常棘手的问题，造成一些问题理论可解而在实际中不一定可解。如 TSP 问题采用穷举法，所有的可行路径共有$\frac{(n-1)!}{2}$条，若以路径比较为基本操作，则需要$\frac{(n-1)!}{2}-1$次比较才能获得最优解。如果以计算机 1 s 可以完成 24 个城市所有路径的枚举为单位，则 25 个城市的枚举需要 24 s。类似地，城市数与计算时间的关系见表 6.1。当城市数目增加到 30 个时，计算

时间约 10.8 年，已经超出可接受的范围。

表 6.1 TSP 问题枚举计算的城市数与计算时间的关系

城市数	24	25	26	27	28	29	30	31
计算时间	1 s	24 s	10 min	4.3 h	约 4.9 d	136.5 d	约 10.8 y	约 325 y

鉴于许多问题的求解复杂度可能大于 TSP 问题，因此只有了解所研究问题和算法的复杂度，才能有针对性地设计和改进算法，提高算法的优化效率。

1. 复杂度的基本概念

基本蚁群算法的复杂度分析是在理论上对蚁群算法的算法效率的分析。可分为算法的时间复杂度分析和空间复杂度分析。

定义 6.1（算法的时间复杂度）：指求解该问题的所有算法中时间复杂性最小的算法时间复杂度。

在实际中，衡量一个算法的好坏通常是用算法中所使用的加、减、乘、除和比较等基本运算的总次数以及具体的问题在计算机计算时的二进制输入数据的大小关系来度量。由于对一个问题的二进制输入长度和算法的基本计算总次数是粗略估计的，一般总是给出一个上限。

定义 6.2（算法的空间复杂度）：指求解该问题的所有算法中空间复杂性最小的算法空间复杂度。

在实际中，通常把算法执行时间内所占用的存储单元定义为算法的空间复杂度。由于基本蚁群算法的空间复杂度非常简单，不做单独讨论，详见文献 [20]。

下面分析基本蚁群算法的时间复杂度。首先定义数量级的概念。

定义 6.3（数量级）：给定自然数 n 的两个函数 $F(n)$ 和 $G(n)$，当且仅当存在一个正常数 K 和一个 n_0，使得当 $n \geqslant n_0$ 时，有 $F(n) \leqslant KG(n)$，则称函数 $F(n)$ 以函数 $G(n)$ 为界，记作 $F(n)=O(G(n))$，或称 $F(n)$ 是 $O(G(n))$。此处的"O"表示数量级的概念。

基本蚁群算法的问题规模可表示为 n 的函数，其中时间复杂度记为 $T(n)$。

2. 基本蚁群算法的时间复杂度分析

对于文中的基本蚁群算法，n 为 TSP 的规模，m 为人工蚂蚁数量，N_c 为算法的循环次数，从蚁周模型的实现过程可以看到该算法各环节的时间复杂度，见表 6.2。

表 6.2 基本蚁群算法时间复杂度分析表

Step	内容	$T(n)$

1	初始化
	set $t=0$;set $N_c=0$;//t 为时间计数器,N_c 为循环计数器
	set $\tau_{ij}(t)=const$,$\Delta\tau_{ij}=0$;// 设置信息素初值
	将 m 个蚂蚁随机置于 n 个节点上
		$O(n^2+m)$

2	设置蚂蚁禁忌表 set $s=0$;//s 为禁忌表指针 for $k=1$ to m do 置第 k 只蚂蚁的起始城市到禁忌表 $tabu_k(s)$	$O(m)$
3	每只蚂蚁单独构造解 循环计算直到禁忌表满 // 共需循环($n-1$) 次 set $s=s+1$; for $k=1$ to m do 根据转移概率 $p_{ij}^k(t)$ 选择下一城市 将城市序号 j 加入禁忌表 $tabu_k(s)$	$O(n^2m)$
4	解的评价和轨迹更新量的计算 for $k=1$ to m do 将第 k 只蚂蚁从禁忌表 $tabu_k(n)$ 移到 $tabu_k(1)$ 计算第 k 只蚂蚁在本次循环中的路径长度 更新最优路径 计算各条路径上的信息素反馈量 $\Delta\tau_{ij}$	$O(n^2m)$
5	信息素轨迹浓度的更新 计算各条路径上在下一轮循环开始前的信息素强度 $\tau_{ij}(t+n)$ set $t=t+n$;set $N_c=N_c+1$	$O(n^2)$
6	判定是否达到终止条件 如果 $N_c \leqslant N_{cmax}$,且搜索没有出现停止现象 清空全部禁忌表 返回 step2 否则 打印最短路径 结束	$O(nm)$

Dorigo 等曾对经典的 TSP 问题求解复杂度进行了深入研究，所得到的结果表明，算法的时间复杂度为 $T(n)=O(N_c \cdot n^2 \cdot m)$；当参与搜索的蚂蚁个数 m 大致与问题规模 n 相等时，算法的时间复杂度为 $T(n)=O(N_c \cdot n^3)$。

6.2.5 参数选择对蚁群算法性能的影响

探索（Exploration）和开发（Exploitation）能力的平衡是影响算法性能的一个重要方面，也是蚁群算法研究的关键问题之一。探索能力是指蚁群算法要在解空间中测试不同区域以找到一个局优解的能力；开发能力是指蚁群算法在一个有希望的区域内进行精确搜索的能力。那么该如何设定蚁群算法中的各种参数，实现探索和开发能力的平衡呢？这里，探索与开发实际上就是前面章节所说的全局搜索能力和局域搜索能力。

由于蚁群算法参数空间的庞大性和各参数之间的关联性，很难确定最优组合参数使蚁群算法求解性能最佳，至今还没有完善的理论依据。大多数情况下是通过试验的反复试凑得到的。目前已经公布的蚁群算法参数设置成果都是就特定问题所采用的特定蚁群算法而言，以应用最多的蚁周模型（Ant-Cycle）为例，其最好的试验结果为

$$0 \leqslant \alpha \leqslant 5,\ 0 \leqslant \beta \leqslant 5,\ 0.1 \leqslant \rho \leqslant 0.99,\ 10 \leqslant Q \leqslant 10\ 000$$

那么到底有没有确定最优组合参数的一般方法呢？为了回答这个问题，先分析以下参数对蚁群算法性能的影响。

（1）信息素和启发函数对蚁群算法性能的影响

信息素 τ_{ij} 是表征过去信息的载体，而启发函数 η_{ij} 则是表征未来信息的载体，它们直接影响到蚁群算法的全局收敛性和求解效率。

（2）信息素残留因子对蚁群算法性能的影响

参数 ρ 表示信息素挥发因子，其大小直接关系到蚁群算法的全局搜索能力及其收敛速度；参数 $1-\rho$ 表示信息素残留因子，反映了蚂蚁个体之间相互影响的强弱。信息素残留因子 $1-\rho$ 的大小对蚁群算法的收敛性能影响非常大。在 0.1～0.99 范围内，$1-\rho$ 与迭代次数 N_c 近似成正比，这是由于 $1-\rho$ 很大，路径上的残留信息占主导地位，信息正反馈作用相对较弱，搜索的随机性增强，因而蚁群算法的收敛速度很慢。若 $1-\rho$ 较小时，正反馈作用占主导地位，搜索的随机性减弱，导致收敛速度快，但易于陷于局优状态。

（3）蚂蚁数目对蚁群算法性能的影响

蚁群算法是通过多个候选解组成的群体进化过程来搜索最优解，所

以蚂蚁的数目 m 对蚁群算法有一定影响。蚂蚁数量 m 大（相对处理问题的规模），会提高蚁群算法的全局搜索能力和稳定性，但数量过大会导致大量曾被搜索过的路径上的信息量变化趋于平均，信息正反馈作用减弱，随机性增强，收敛速度减慢。反之，蚂蚁数量 m 小（相对处理问题的规模），会使从来未被搜索到的解上的信息量减小到接近于 0，全局搜索的随机性减弱，虽然收敛速度加快，但会使算法的稳定性变差，出现过早停滞现象。经大量的仿真试验获得：当城市规模大致是蚂蚁数目的 1.5 倍时，蚁群算法的全局收敛性和收敛速度都比较好。

（4）启发式因子、期望启发式因子、信息素强度对蚁群算法性能的影响

启发式因子 α 反映蚂蚁在运动过程中所积累的信息量在指导蚁群搜索中的相对重要程度。α 越大，蚂蚁选择以前走过路径的可能性就越大，搜索的随机性减弱；α 越小，易使蚁群算法过早陷入局优。

期望启发式因子 β 反映了启发式信息在指导蚁群搜索过程中的相对重要程度，这些启发式信息表现为寻优过程中先验性、确定性因素。β 越大，蚂蚁在局部点上选择局部最短路径的可能性越大，虽然加快了收敛速度，但减弱了随机性，易于陷入局部最优。

信息素强度 Q 为蚂蚁循环一周时释放在所经路径上的信息素总量。Q 越大，蚂蚁在已遍历路径上信息素的累积越快，加强蚁群搜索时的正反馈性，有助于算法的快速收敛。

基于以上各种参数对算法收敛性的影响，段海滨提出了设定蚁群算法参数“三步走”的思想（参见文献［20］）。其步骤如下：

第 1 步：确定蚂蚁数目，确定原则：城市规模/蚂蚁数目 ≈ 1.5。

第 2 步：参数粗调，调整取值范围较大的信息启发式因子 α、期望启发式因子 β 以及信息素强度 Q 等参数，以得到较理想的解。

第 3 步：参数微调，调整取值范围较小的信息素挥发因子 ρ。

6.3 改进的蚁群算法

虽然与已经发展完备的一些启发式算法比较起来，基本蚁群算法的计算量比较大，搜索时间长，但在解决某些问题时，蚁群算法有很大的优越性，尤其是 TSP 问题。它的成功运用吸引了国际学术界的普遍关注，并提出了各种有益的改进算法。在了解这些改进研究之前，先了解一下基本蚁群算法的不足。

（1）每次解构造过程的计算量较大，算法搜索时间较长。算法的

计算复杂度主要在解构造过程，比如 TSP 问题时间复杂度为 $O(N_c \cdot n^2 \cdot m)$。

（2）算法容易出现停滞现象，即搜索进行到一定的程度后，所有蚂蚁搜索到的解完全一致，不能对空间进一步进行搜索，不利于发现更好的解。

（3）基本蚁群优化算法本质上是离散的，只适用于组合优化问题，对于连续优化问题（函数优化）无法直接应用，限制了算法的应用范围。

针对蚁群算法的缺陷，蚁群算法的改进研究主要目的有两点。一是在合理的时间内提高蚁群算法的寻优能力，改善其全局收敛性；二是使其能够应用于连续域问题。

在介绍蚁群算法改进研究之前，先了解蚁群算法收敛性研究的成果。因为收敛性研究可以为改进蚁群算法提供理论依据和指导。

6.3.1 蚁群算法的收敛性研究

所有的仿生优化算法都要考虑收敛性问题，蚁群算法也不例外。虽然蚁群算法已经有多种不同版本的改进算法并成功应用于诸多领域，但大部分是经验性的实验研究，缺乏必要的理论框架及相应的理论基础和依据，只有少部分改进的蚁群算法给出了收敛性证明，这在很大程度上阻碍了蚁群算法的发展。

最先开始蚁群算法收敛性研究的是 Gutjahr W J，他从有向图论的角度对一种改进的蚁群算法 GBAS（Graph-based Ant System）的收敛性进行了证明；Stützle T 和 Dorigo M 针对具有组合优化性质的极小化问题提出了一类改进蚁群算法 $ACO_{gb,\tau_{\min}}$，并对其收敛性进行了理论分析；针对上述两种改进蚁群算法中的一些缺陷，根据所采用的信息素更新规则的不同，Gutjahr W J 又提出两种新的 GBAS，即时变信息素挥发系数 GBAS/TDEV 算法（GBAS with Time-Dependent Evaporation Factor）和时变信息素下界 GBAS/TDLB 算法（GBAS with Time-Dependent Lower Pheromone Bound），证明了可通过选择合适的参数来保证蚁群算法的动态随机过程收敛到全局最优解；Hou Y H 等对一类广义蚁群算法（Generalized Ant Colony Algorithm，GACA）进行的基于不动点理论的收敛性分析；Yoo J H 等对一类分布式蚂蚁路由随机算法的收敛性研究；Badr A 等将蚁群算法模型转化为分支随机过程，从分支随机路径和分支 Wiener 过程的角度推导了蚂蚁路径存亡的比率，并证明了该过程为稳态分布；孙焘等对一类简单蚁群算法的收敛性及有关参数问题做了初步

研究；丁建立等对一种遗传－蚁群算法的收敛性进行了 Markov 理论分析，并证明其优化解满意值序列单调不增且收敛；段海滨等对基本蚁群算法进行 Markov 理论分析，运用离散鞅（Discrete Martingale）研究工具，提出蚁群算法首达时间的定义，同时对蚁群算法首次到达时间的期望值作了初步分析。

算法的收敛性研究不仅对深入理解算法机理具有重要的理论意义，而且对改进算法、编写算法程序具有非常重要的现实指导意义。蚁群算法的收敛性研究是一个非常重要的研究内容。

6.3.2 离散域蚁群算法的改进研究

国内外，离散域蚁群算法的改进研究成果很多，如自适应蚁群算法、基于信息素扩散的蚁群算法、基于去交叉局部优化策略的蚁群算法、多态蚁群算法、基于模式学习的小窗口蚁群算法、基于混合行为蚁群算法、带聚类处理的蚁群算法、基于云模型理论的蚁群算法、具有感觉和知觉特征的蚁群算法、具有随机扰动特性的蚁群算法、基于信息熵的改进蚁群算法等。这里不能一一列举，仅介绍离散域优化问题的自适应蚁群算法。

什么是自适应蚁群算法？即对蚁群算法的状态转移概率、信息素挥发因子、信息量等因素采用自适应调节策略为一种基本改进思路的蚁群算法。下面介绍自适应蚁群算法中两个最经典的方法，一个是蚁群系统（Ant Colony System，ACS)，另一个是最大－最小蚁群系统（MAX-MIN Ant System，MMAS)。

一、蚁群系统（Ant Colony System，ACS)

蚁群系统（Ant Colony System，ACS）模型最早是由 Dorigo，Gambardella 等人在基本蚁群算法（AS）的基础上提出的。下面介绍 ACS 蚁群系统模型的构成和算法。

ACS 解决了基本蚁群算法在构造解过程中，随机选择策略造成的算法进化速度慢的缺点。该算法在每一次循环中仅让最短路径上的信息量作更新，且以较大的概率让信息量最大的路径被选中，充分利用学习机制，强化最优信息的反馈。ACS 的核心思想是：蚂蚁在寻找最佳路径的过程中只能使用局部信息，即采用局部信息对路径上的信息量进行调整；在所有进行寻优的蚂蚁结束路径的搜索后，路径上的信息量会再一次调整，这次采用的是全局信息，而且只对过程中发现的最好路径上的信息量进行加强。ACS 模型与 AS 模型的主要区别有 3 点：①蚂蚁的状态转移规则不同；②全局更新规则不同；③新增了对各条路径信息量调

整的局部更新规则。下面展开介绍。

1. ACS 的状态转移规则

为了避免停滞现象的出现，ACS 采用了确定性选择和随机性选择相结合的选择策略，并在搜索过程中动态调整状态转移概率。即对位于城市 i 的蚂蚁 k 按照式（6.9）选择下一个城市：

$$j=\begin{cases}\arg\max\limits_{s\in J_k(i)}\{[\tau(i,s)]^{\alpha}[\eta(i,s)]^{\beta}\}, & q\leqslant q_0\\ \text{式}(6.10), & \text{其他}\end{cases} \tag{6.9}$$

其中，$J_k(i)$ 是第 k 只蚂蚁在访问到城市 i 后尚需访问的城市集合，q 为一个在区间［0，1］内的随机数，q_0 是一个算法参数（$0\leqslant q_0\leqslant 1$）；当 $q>q_0$ 时，蚂蚁 k 根据式（6.10）确定由城市 i 向下转移的目标城市：

$$p_{ij}^{k}=\begin{cases}\dfrac{[\tau(i,j)]^{\alpha}\cdot[\eta(i,j)]^{\beta}}{\sum\limits_{s\in J_k(i)}[\tau(i,s)]^{\alpha}\cdot[\eta(i,s)]^{\beta}}, & j\in allowed_k\\ 0, & \text{其他}\end{cases} \tag{6.10}$$

式（6.9）所确定的蚂蚁转移到下一个城市的方法称为自适应伪随机概率选择规则（Pseudo-Random Proportional Rule）。在这种规则下，每当蚂蚁要选择向哪一个城市转移时，就产生一个在［0，1］范围内的随机数，根据这个随机数的大小按公式（6.9）确定用哪种方法产生蚂蚁转移的方向。

2. ACS 全局更新规则

在 ACS 蚁群算法中，全局更新不再用于所有的蚂蚁，而是只对每一次循环中最优的蚂蚁使用。更新规则如下式：

$$\tau(i,j)\leftarrow(1-\rho)\cdot\tau(i,j)+\rho\cdot\Delta\tau(i,j) \tag{6.11}$$

且

$$\Delta\tau(i,j)=\begin{cases}1/L_{gb}, & (i,j)\text{为全局最优路径且 }L_{gb}\text{是最短路径}\\ 0, & \text{其他}\end{cases} \tag{6.12}$$

其中，L_{gb}为蚁群当前循环中所求得的最优路径长度；ρ 为一个（0，1）区间的参数，其意义相当于蚁群算法基本模型中路径上的信息素挥发系数。

3. ACS 局部更新规则

局部更新规则是在所有的蚂蚁完成一次转移后执行

$$\tau(i,j)\leftarrow(1-\rho)\cdot\tau(i,j)+\rho\cdot\Delta\tau(i,j) \tag{6.13}$$

其中，ρ 为一个（0，1）区间的参数，其意义也相当于蚁群算法基本模型中路径上的信息素挥发系数。$\Delta\tau(i,j)$ 的取值方法有下列三种方案

①$\Delta\tau(i,j)=0$；

②$\Delta\tau(i,j)=\tau_0$，τ_0为路径上信息量的初始值；

③$\Delta\tau(i,j)=\gamma\cdot\max\limits_{z\in J_k(j)}\tau(j,z)$，其中的$J_k(j)$表示第$k$只蚂蚁在访问到城市$j$后尚需访问的城市集合。

上述采用$\Delta\tau$（i，j）取值的第③种方案的ACS算法被称为Ant－Q强化学习的蚁群算法。实验结果表明与AS基本蚁群算法相比，Ant－Q system模型具有一般性，而且更有利于全局搜索。

算法的实现过程可以用以下的伪代码来表示

```
begin
初始化过程：
    ncycle = 1；
    bestcycle = 1；
    Δτij(i, j) = τ0 = C； α； β； ρ； q0；
    ηij(由某种启发式算法确定)；
    tabuk = ∅；
while(not termination condition)
    {for(k = 1； k < m； k + +)
        {将 m 个蚂蚁随机放置于初始城市上；}
    for(index = 0； index < n； index + +) (index 为当前循环中已
    经走过的城市个数)
      {for(k = 0； k < m； k + +)
          {产生随机数 q
           按公式 (6.9) 和式 (6.10) 规则确定每只蚂蚁将要
          转移的位置；
           将刚刚选择的城市 j 加入到 tabuk 中；
           按公式 (6.13) 执行局部更新规则；
          }
      }
    确定本次循环中找到的最佳路径 L = min(Lk)， k = 1， 2， …， m；
    根据式 (6.11) 和式 (6.12) 执行全局更新规则；
      ncycle = ncycle + 1；
    }
输出最佳路径及结果；
end
```

二、最大－最小蚂蚁系统（MAX－MIN Ant System，MMAS）

通过对蚁群系统的研究表明，将蚂蚁搜索行为集中到最优解的附近可以提高解的质量和收敛速度，从而改进算法的性能。但是这种搜索方式会使算法过早收敛而出现早熟现象。针对这个问题，德国学者 Stützle T 和 Hoos H H 提出的最大 - 最小蚂蚁系统（MAX - MIN ant system，MMAS）。

MMAS 的基本思想：仅让每一代中的最好个体所走路径上的信息量作调整，从而更好地利用了历史信息，以加快收敛速度。但这样更容易出现过早收敛的停滞现象，为了避免算法过早收敛于非全局最优解，将各条路径上的信息量限制在区间 $[\tau_{min}, \tau_{max}]$ 之内，超出这个范围的值将被限制为信息量允许值的上下限，这样可以有效地避免某条路径上的信息量远大于其他路径而造成的所有蚂蚁都集中到同一条路径上，从而使算法不再扩散，加快收敛速度。MMAS 是解决 TSP、QAP 等离散域优化问题的最好蚁群算法之一，很多对蚁群算法的改进算法都渗透着 MMAS 的思想。

MMAS 蚁群算法在基本蚁群算法（AS）的基础上作了三点改进：

（1）首先初始化信息量 $\tau_{ij}(t)=c$ 设为最大值 τ_{max}。

（2）其次各个蚂蚁在一次循环后，只有找到最短路径的蚂蚁才能够在其经过的路径上释放信息素。即

$$\tau_{ij}(t+n)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij}^{min} \tag{6.14}$$

$$\Delta\tau_{ij}^{min}=Q/L, L=\min(L_k), k=1,2,\cdots,m \tag{6.15}$$

（3）最后将 $\tau_{ij}(t)$ 限定在 $[\tau_{min}, \tau_{max}]$ 之间。如果 $\tau_{ij}(t)<\tau_{min}$，则 $\tau_{ij}(t)=\tau_{min}$；如果 $\tau_{ij}(t)>\tau_{max}$，则 $\tau_{ij}(t)=\tau_{max}$。

算法的实现过程可以用以下的伪代码来表示：

```
begin
初始化过程：
    ncycle = 1;
    bestcycle = 1;
    τmax; τmin; τij = τmax; Δτij = 0;
    ηij(由某种启发式算法确定);
    tabuk = ∅;
while(not termination condition)
    {for(k = 1; k < m; k ++)
        {将 m 个蚂蚁随机放置于初始城市上;}
    for(index = 0; index < n; index ++) (index 为当前循环中已
经走过的城市个数)
```

{for($k=0$; $k<m$; k ++)

{以概率 $p_{ij}^{k}(t)$ 选择下一个城市 j, $j \in allowed_k(t)$;

将刚刚选择的城市 j 加入到 $tabu_k$ 中;

}

}

$ncycle = ncycle + 1$;

确定本次循环中找到的最佳路径 $L=\min(L_k)$, $k=1, 2, \cdots, m$;

根据公式 (6.14)、(6.15) 计算 $\Delta\tau_{ij}^{\min}(ncycle)$, $\tau_{ij}(ncycle+1)$;

如果 $\tau_{ij}(t)<\tau_{\min}$,则 $\tau_{ij}(t)=\tau_{\min}$;

如果 $\tau_{ij}(t)>\tau_{\max}$,则 $\tau_{ij}(t)=\tau_{\max}$;

}

输出最佳路径及结果;

end

除了以上两种自适应改进蚁群算法外，还有很多离散域改进蚁群算法。虽然这些改进策略的侧重点和改进形式不同，但是其目的是相同的，即避免陷入局优，缩短搜索时间，提高蚁群算法的全局收敛性能。

6.3.3 连续域蚁群算法的改进研究

很多工程上的实际问题通常表达为一个连续的最优化问题，并随着问题规模的增大以及问题本身的复杂度增加，对优化算法的求解性能提出越来越高的要求。而基本蚁群算法优良高效的全局优化性能却只能适用于离散的组合优化问题。因为基本蚁群算法的信息量留存、增减和最优解的选取都是通过离散的点状分布求解方式来进行的，所以基本蚁群算法从本质上只适合离散域组合优化问题，离散性的本质限制了其在连续优化领域中的应用。在连续域优化问题的求解中，其解空间是一种区域性的表示方式，而不是以离散的点集来表示的。因此，将基本蚁群算法寻优策略应用于连续空间的优化问题需要解决以下三点。

1. 调整信息素的表示、分布及存在方式

是至关重要的一点，在组合优化问题中，信息素存在于目标问题离散的状态空间中相邻的两个状态点之间的连接上，蚂蚁在经过两点之间的连接的时候释放信息素，影响其他蚂蚁，从而实现一种分布式的正反馈机制，每一步求解过程中的蚁群信息素留存方式只是针对离散的点或点集分量；而用于连续域寻优问题的蚁群算法，定义域中每个点都是问题的可行解，不能直接将问题的解表示成为一个点序列，显然也不存在点间的连接，只能根据目标函数值来修正信息量，在求解过程中，信息

素物质则是遗留在蚂蚁所走过的每个节点上，每一步求解过程中的信息素留存方式在对当前蚁群所处点集产生影响的同时，对这些点的周围区域也产生相应的影响。

2. 改变蚁群的寻优方式

由于连续域问题求解的蚁群信息留存及影响范围是区间性的，非点状分布，所以在连续域寻优过程中，不但要考虑蚂蚁个体当前位置所对应的信息量，还要考虑蚂蚁个体当前位置所对应特定区间内的信息量累计与总体信息量的比较值。

3. 改变蚁群的行进方式

将蚁群在离散解空间点集之间跳变的行进方式变为在连续解空间中微调式的行进方式，这一点较为容易。

近年来，随着蚁群算法的不断发展，拓展蚁群优化算法的功能，使之适用于连续问题已经有了一些成果，分为以下三类。

（1）将蚁群优化框架与进化算法结合，从而实现连续优化算法。

第一个连续蚁群算法就是由 Bilchev G A 等人基于这种思路构建的，求解问题时先使用遗传算法对解空间进行全局搜索，然后利用蚁群算法对所得结果进行局部优化，但该种算法在运行过程中常会出现蚂蚁对同一个区域进行多次搜索的情况，降低了算法的效率；杨勇等人提出了一种求解连续域优化问题的嵌入确定性搜索蚁群算法，该算法在全局搜索过程中，利用信息素强度和启发式函数确定蚂蚁移动方向，而在局部搜索过程中嵌入了确定性搜索，以改善寻优性能，加快收敛速度。

（2）将连续空间离散化，从而将原问题转化为一个离散优化问题，然后应用基本蚁群优化算法的原理来求解。

高尚等人提出了一种基于网格划分策略的连续域蚁群算法；汪镭等人提出的基于信息量分布函数的连续域蚁群算法；李艳君等人提出了一种用于连续域优化问题求解的自适应蚁群算法；段海滨等人提出一种基于网格划分策略的自适应连续域蚁群算法；陈崚等人提出的一种基于交叉变异操作的连续域蚁群算法等。采用这种方式来研究的比较多，但当问题规模增大时，经离散化后，问题的求解空间将急剧增大，寻优难度将大大增加。对于较大规模的连续优化问题这类方法的适应性还有待进一步的验证。

（3）对蚂蚁行为模型进行更加深入的、广泛的研究，从而构造新的蚁群算法，应用于连续问题的求解。

Dréo J 等人提出了一种基于密集非递阶的连续交互式蚁群算法 CIACA，该算法通过修改信息素的留存方式和行走规则，并运用信息素

交流和直接通信两种方式来指导蚂蚁寻优；Pourtakdoust S H 等人提出了一种仅依赖信息素的连续域蚁群算法；张勇德等人提出了一种用于求解带有约束条件的多目标函数优化问题的连续域蚁群算法。

由于篇幅所限，这里不能一一列举，仅介绍两种，即（1）中，杨勇等人提出的嵌入确定性搜索的连续域蚁群算法；（3）中，Dréo J 等人提出的基于密集非递阶的连续交互式蚁群算法 CIACA。

1. 嵌入确定性搜索的连续域蚁群算法

嵌入确定性搜索的连续域蚁群算法在全局搜索过程中，利用信息素强度和启发式函数确定蚂蚁的移动方向；而在局部搜索过程中，嵌入了确定性搜索，以改善寻优性能，加快收敛速率。

设优化函数为 $\max Z=f(X)$，m 只蚂蚁随机分布在定义域内，每只蚂蚁都有一个邻域，其半径为 r。每只蚂蚁在自己的领域内进行搜索，当所有蚂蚁完成局部搜索后，蚂蚁个体根据信息素强度和启发式函数在全局范围内进行移动，完成一次循环后，则进行信息素强度的更新计算。

（1）局部搜索

局部搜索是指每只蚂蚁在自己的邻域空间内进行随机搜索。设新的位置点为 X'，如果新的位置值比原来目标函数值大，则取新位置，否则舍去。局部搜索是在半径为 r 的区域内进行的，且 r 随迭代次数的增加而减少。有

$$X_i=\begin{cases}X_i', & f(X_i')>f(X_i)\\ X_i, & \text{其他}\end{cases} \tag{6.16}$$

（2）全局搜索

全局搜索是指每只蚂蚁都经过一次局部搜索后，选择停留在原地、转移到其他蚂蚁的邻域或进行全局随机搜索。设 $Act(i)$ 为第 i 只蚂蚁选择的动作，f_{avg} 为 m 只蚂蚁的目标函数平均值，则有

$$Act(i)=\begin{cases}\text{全局随机搜索}, & f(X_i)<f_{avg}\cap q<q_0\\ S, & \text{其他}\end{cases} \tag{6.17}$$

其中，q 为一个在区间 [0, 1] 内的随机数；q_0 是一个算法参数（$0\leqslant q_0\leqslant 1$）；$S$ 按如下转移规则选择动作：

$$p(i,j)=\frac{\tau(j)\mathrm{e}^{-\frac{d_{ij}}{T}}}{\sum\tau(j)\mathrm{e}^{-\frac{d_{ij}}{T}}} \tag{6.18}$$

其中，$d_{ij}=f(X_i)-f(X_j)$，且当 $i\neq j$ 时，$d_{ij}<0$；而当 $i=j$ 时，$d_{ij}=0$。式（6.18）保证了第 i 只蚂蚁按概率向其他目标函数值更大的蚂蚁 j 的

邻域移动，其中系数 T 的大小决定了这个概率函数的斜率。

蚂蚁向某个信息素强度高的地方移动时，可能会在转移路途中的一个随机地点发现新的食物源，这里将其定义为有向随机转移。第 i 只蚂蚁向第 j 只蚂蚁的邻域转移的公式为

$$X_i = \begin{cases} X_j & \rho < \rho_0 \\ \alpha X_j + (1-\alpha) X_i & \text{其他} \end{cases} \tag{6.19}$$

其中，$0 < \rho < 1$；$\rho_0 > 0$；$\alpha < 1$。

（3）信息素强度更新规则

全局搜索结束后，要对信息素强度进行更新。更新规则为：如果有 n 只蚂蚁向蚂蚁 j 处移动（包括有向随机搜索），则有

$$\tau(j) = \beta\tau(j) + \sum_{i=1}^{n} \Delta\tau_i \tag{6.20}$$

其中，$\Delta\tau_i = \dfrac{1}{f(X_i)}$；$0 < \beta < 1$ 是遗忘因子。

以上 3 个步骤模仿了自然界蚂蚁寻食的过程，蚂蚁个体通过局部随机搜索寻找食物源，然后利用信息素交换信息，决定全局转移方向。全局随机搜索的蚂蚁承担搜索陌生新食物源的任务，本质上也是一种随机性搜索算法。

（4）嵌入确定性搜索

随机性搜索算法存在着求解效率较低、求解结果较分散等缺点，因此有必要引入确定性搜索，对其加以改进。这里考虑使用确定性搜索中的直接法，直接法只利用函数信息而不需要利用导数信息，甚至不要求函数连续，适用面较广，易于编程，避免复杂计算。常用的直接法包括网格法、模式搜索法、二坐标轮换法等，文中采用了模式搜索法中的步长加速法。

步长加速法是在坐标轮换法的基础上发展起来的，包括探测性搜索和模式性移动两部分。首先依次沿坐标方向探索，称之为探测性搜索；然后经此探测后求得目标函数的变化规律，从而确定搜索方向并沿此方向移动，称之为模式移动。重复以上两步，直到探测步长小于充分小的正数 ε 为止，具体步骤参见文献［3］。

嵌入确定性搜索的蚁群算法，是在局部搜索时以一定的概率利用步长加速法进行确定性搜索。局部搜索规则如下：

$$R = \begin{cases} \text{用步长加速法进行局部确定性搜索，} & v < v_0 \\ \text{按公式（6.16）进行局部随机搜索，} & \text{其他} \end{cases} \tag{6.21}$$

其中，v 是随机数且 $0 < v < 1$；v_0 是算法系数且 $0 < v_0 < 1$。

嵌入确定性搜索的蚁群算法的具体步骤如下

初始化；

Loop；

　　每只蚂蚁处于每次循环的开始位置；

　　Loop；

　　　每只蚂蚁利用式（6.21）进行局部搜索；

　　Until 所有蚂蚁完成局部搜索；

　　Loop；

　　每只蚂蚁进行全局搜索，按式(6.17)～(6.19)选择要进行的动作；

　　Until 所有蚂蚁完成全局搜索；

按式（6.20）进行信息素强度更新；

Until 中止条件。

2. 基于密集非递阶的连续交互式蚁群算法（CIACA）

基于密集非递阶的连续交互式蚁群算法（Continuous Interacting Ant Colony Algorithm，CIACA）的思想源于对自然界中真实蚁群行为和求解连续域优化问题蚁群算法机理的进一步研究。该算法通过修改蚂蚁信息素的留存方式和蚂蚁的行走规则，并运用信息素交流和直接通信两种方式来指导蚂蚁寻优。CIACA 是一种崭新的蚁群算法。在介绍 CIACA 之前，先了解一下密集非递阶的生物学概念。

（1）密集非递阶的概念和简单的非递阶算法

① 密集非递阶的概念

“密集非递阶（Dense Heterarchy）”最早由 Wilson E D 于 1988 年提出。“蚁群是一个特殊的层次结构，可称之为非递阶结构。这意味着较高层次单元的性质在一定程度上影响着较低的一层，而被较高层次影响后的较低层次单元会反过来影响较高层次”，这一思想提出了两种通信通道，即基于信息素轨迹交流通信通道和蚂蚁个体间直接通信通道，这两种通道对于蚁群算法非常重要。“密集非递阶”用于描述蚁群从环境中接受“信息流”方式的一个基本概念，每只蚂蚁都可在任意时刻与其他蚂蚁进行联络，而蚁群中的信息流是通过多个通信通道传输的。

为了形象说明密集非递阶结构与层次结构的不同，参考图 6.4。层次结构是一种金字塔形的结构，就像是部队中军长传令给师长，师长传令给旅长，其余以此类推。而密集非递阶结构中，“蚁后”并不传令给其他蚂蚁，而是作为蚁群网络中的普通一员，这种没有“层次”的系统具有很强的自组织功能。

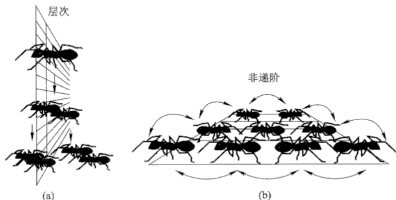


图 6.4 层次结构与非递阶结构示意图

② 简单的非递阶算法

这里先介绍一个简单的非递阶算法，该算法利用了通信通道的基本思想，一个通信通道是信息素的存放地，可用来传递多种信息，如图 6.5 所示。

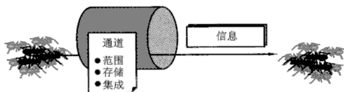


图 6.5 信息通道示意图

信息通道的基本性质如下。

a. 范围：即蚁群中信息素的交流方式，蚁群中的某一子群可与另一子群进行信息交流。

b. 存储：即信息素在系统中的驻留方式，信息素可在某一时间段内被一直保留。

c. 集成：即信息素在系统中的进化方式，信息素可通过一个外部过程被一只或多只蚂蚁更新，也可不更新。

上述性质都集聚于同一信息通道，这样就形成了许多不同种类的信息通道。蚂蚁通信中所传递的信息具有多种形式，有时很难描述某些特殊类别的信息。

(2) CIACA 通信通道

按照采用通信通道的不同，定义了三种版本的 CIACA。即信息素交流的 CIACA，利用个体之间的直接通信的 CIACA 和二者协同的 CIACA。

① 信息素交流的 CIACA

第一个版本的 CIACA 与 Bilchev G A 等率先提出的用于求解连续域优化问题的改进蚁群算法很接近。该算法受蚂蚁的信息素存留启发而设置了一个通信通道，每只蚂蚁在其搜索空间内的某一节点上释放一定量的信息素，节点上的信息量与其所搜索到的目标函数值成正比。这些信息节点能够被蚁群中的所有个体察觉，并逐渐消失。蚂蚁根据路径距离和路径上的信息量来决定是否选择这些信息节点。蚂蚁会向着信息素点集云的重心 G_j 移动，而重心位置依赖于第 i 个节点上第 j 只蚂蚁的“兴趣” ω_{ij}，表示如下：

$$G_j = \sum_{i=1}^{n} \left(\frac{x_i \omega_{ij}}{\sum_{i=1}^{n} \omega_{ij}} \right) \tag{6.22}$$

$$\omega_{ij} = \frac{\bar{\delta}}{2} \cdot e^{-\theta_i \cdot \delta_{ij}} \tag{6.23}$$

其中，n 表示节点数目；x_i 表示第 i 个节点的位置；$\bar{\delta}$ 表示蚁群中两只蚂蚁间的平均距离；θ_i 表示第 i 个节点上的信息量；δ_{ij}表示从第 j 只蚂蚁到第 i 个节点之间的距离。值得注意的是，处于信息素节点上的蚂蚁并不径直地向信息素点集云的重心移动。事实上，每只蚂蚁都有在蚁群中均匀分布的参数调整范围，每只蚂蚁都得到一个允许范围内的随机距离，蚂蚁会以随机距离为度量向着其重心位置移动，但是某些干扰因素可能会影响蚂蚁所到达的最终位置。

从非递阶概念的角度来描述上述行为，该 CIACA 中信息素交流通道的性质如下。

a. 范围：当蚁群中某只蚂蚁留下一定量的信息素后，其他后继蚂蚁都能觉察到该信息素的存在。

b. 存储：某一时间段内信息素将被一直保留于蚁群系统之中。

c. 集成：由于信息素的挥发作用，随着时间的推移信息素将被更新。

② 利用个体之间的直接通信的 CIACA

每只蚂蚁都能给另一只蚂蚁发送“消息”，这意味着该通信通道的范围是“点对点”式的。蚂蚁可将已经接收到或将要接收到的信息存储到栈中，而栈中的信息可被随机读取。此处所发送的“消息”是信息发送者的位置，即目标函数值。信息接收者会将发送者所发送来的“消息”与其自身的信息相比较，以决定它是否要向信息发送者的位置移动。最终位置将出现在一个以信息发送者为中心、信息接收者

范围为半径的超球体内，然后信息接收者将“消息”进行压缩并将其随机发送给另一只蚂蚁。此时，该 CIACA 中的信息通道具有如下性质。

a. 范围：当蚁群中的某只蚂蚁发出“消息”后，仅有一只蚂蚁可以觉察到此“消息”。

b. 存储：某一时间段内信息可以以“记忆”的形式保存在蚁群系统中。

c. 集成：所存储的信息是静态的。

③ 二者协同的 CIACA

信息素交流的 CIACA 和利用个体之间的直接通信的 CIACA 具有很大的不同，自组织的作用可将较低层次的个体整合成较高层次的整体。基于这一思想，可将上述两种版本的 CIACA 算法中的简单通信通道融合于一个系统中，由于通信通道没有并发机制，所以实现起来很容易。

(3) CIACA

CIACA 的程序结构流程如图 6.6 所示，算法步骤主要包括以下 3 步。

第 1 步：设置参数。

第 2 步：算法开始。

第 3 步：若满足结束条件，算法结束。

蚂蚁根据其在通信通道系统中所处理的感知信息进行移动，需要设置 4 个参数。

① $\eta \in [0, +\infty)$：系统中蚂蚁的数目，其值可通过式 (6.24) 获得

$$\eta = \eta_{\max}(1 - e^{-\frac{d}{p}}) + \eta_0 \tag{6.24}$$

其中，d 表示目标函数的维数；$\eta_{\max}$ 表示最大蚂蚁数目，一般设置 $\eta_{\max} = 1\,000$；η_0 表示目标函数维数为 0 时的蚂蚁数目，一般设置 $\eta_0 = 5$；p 表示蚂蚁数目的相对重要性，一般设置 $p = 10$。

② $\sigma \in [0, 1]$：搜索空间度的百分比，用来定义蚂蚁移动范围分布的标准偏差，其经验值为 0.9。

③ $\rho \in [0, 1]$：用来定义信息素的持久性，其经验值为 0.1。

④ $\mu \in [0, +\infty)$：“消息”的初始数目，其值可通过公式 $\mu = \frac{2}{3}\eta$ 获得。

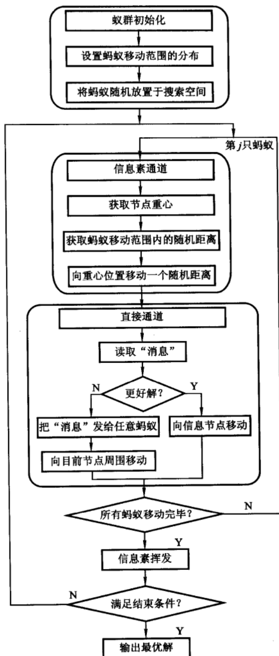


图 6.6 CIACA 的程序结构流程

6.4 蚁群算法与其他仿生优化算法的比较与融合

自从20世纪50年代以来，人们从生物进化的机理中受到启发，构造和设计出许多仿生优化算法，如遗传算法、蚁群算法、粒子群算法、捕食搜索算法等。它们都属于一类模拟自然界生物系统行为或过程的最优化仿生智能算法。它们有着自己的特点，适合不同类型的实际问题，但在某些方面又不谋而合。为了更好地发挥这些仿生优化算法的作用来解决实际问题，学者们将这些算法融合在一起。在介绍这些融合算法之前，先分析这些仿生算法的异同。

6.4.1 蚁群算法与其他仿生优化算法的比较

蚁群算法与遗传算法、粒子群算法、捕食搜索算法都属于仿生优化算法，它们都属于一类模拟自然界生物系统、完全依赖生物体自身本能、通过无意识寻优行为来优化其生存状态以适应环境需要的最优化智能算法。它们有如下相同点。

(1) 都是一类不确定的概率型全局优化算法。仿生优化算法的不确定性是伴随其随机性而来的，其主要步骤含有随机因素，有更多的机会求得全局最优解，比较灵活。

(2) 都不依赖于优化问题本身的严格数学性质，都具有稳健型。在优化过程中都不依赖于优化问题本身的严格数学性质（如连续性、可导性）以及目标函数和约束条件的精确数学描述。因此用仿生优化算法求解许多不同问题时，只需要设计相应的评价函数，而基本上无需修改算法的其他部分。在不同条件和环境下算法的适用性和有效性很强。

(3) 都是一种基于多个智能体的仿生优化算法。仿生优化算法中的各个智能体之间通过相互协作来更好地适应环境，表现出与环境交互的能力。

(4) 都具有本质并行性。仿生优化算法的本质并行性表现在两个方面，一是仿生优化计算的内在并行性，即仿生优化算法本身非常适合大规模并行；二是仿生优化计算的内含并行性，这使得仿生优化算法能以较少的计算获得较大的收益。

(5) 都具有突现性。仿生优化算法总目标的完成是在多个智能体个体行为运动过程中突现出来的。

(6) 都具有自组织性和进化性。在不确定复杂环境中，仿生优化

算法可通过自学习不断提高算法中个体的适应性。

遗传算法、蚁群算法、粒子群算法、捕食搜索算法虽然都属于仿生优化算法，但它们在算法机理、实现形式等方面存在许多不同之处。

（1）遗传算法：以决策变量的编码作为运算对象，借鉴了生物学中的染色体概念，模拟自然界中生物遗传和进化的精英策略，采用个体评价函数进行选择操作，并采用交叉、变异算子产生新的个体，使算法具有较大的灵活性和可扩展性。缺点：求解到一定范围时往往做大量无为的冗余迭代，求精确解效率低。

（2）蚁群算法：采用了正反馈机制或称是一种增强型学习系统，通过不断更新信息素达到最终收敛于最优路径的目的，这是蚁群算法不同于其他仿生优化算法最为显著的特点。缺点：蚁群算法需要较长的搜索时间，且容易出现停滞现象，且该算法的收敛性能对初始化参数的设置比较敏感。

（3）粒子群算法：是一种简单容易实现又具有深刻智能背景的启发式算法，与其他仿生优化算法相比，该算法所需代码和参数较少，而且受所求问题维数的影响较小。缺点：粒子群算法的数学基础相对薄弱，缺少深刻的数学理论分析。

（4）捕食搜索算法：不是一种具体的寻优计算方法，本质上是一种平衡局域搜索和全局搜索的策略。捕食搜索的全局搜索负责对解空间进行广度探索，局域搜索负责对较好区域进行深度开发。二者结合起来具有搜索速度快，搜索质量高，有效避免陷入局优等优点。

6.4.2 蚁群算法与其他仿生优化算法的融合

蚁群算法具有较强的鲁棒性、优良的分布式计算机制、易于与其他方法结合等优点。利用蚁群算法以上优点，将其与其他仿生优化算法融合，得到很多融合策略，如蚁群算法与遗传算法融合策略、蚁群算法与粒子群算法融合策略等。由于篇幅所限，仅介绍蚁群算法与遗传算法的融合策略。

蚁群算法与遗传算法的融合

蚁群算法与遗传算法相融合是蚁群算法与仿生优化算法相互融合方面研究最早且应用最广的一个尝试。最早开始于 Abbattista F 等提出的将遗传算法（GA）和蚁群算法相融合的改进策略，并在 Oliver30TSP 和 Eilon50TSP 的仿真试验中得到了较好的结果；随后，大量的研究将蚁群算法与 GA 相融合解决离散域和连续域中的多种优化问题，并取得了较好的应用效果。下面分别对离散域和连续域中蚁群算法与遗传算法典型

融合策略作详细介绍。

（1）离散域蚁群遗传算法

Pilat M L 等采用 GA 对蚁群算法中 3 个参数 β，ρ，q_0 进行优化，但对 β，ρ，q_0 的取值是相对于 α 的一个比值，α 取默认值，所以无法实现对蚁群算法 4 个组合参数 α，β，ρ，q_0 的全局寻优，也就很难求得 TSP 的全局最优解。针对此问题，孙力娟等人研究了一种求解离散域优化问题的蚁群遗传算法（Ant Colony Algorithm Genetic Algorithm，ACAGA），其核心是应用 GA 对蚁群算法的 4 个参数（α，β，ρ，q_0）进行优化，并运用 MMAS 改进蚁群算法的寻径行为，以实现对搜索空间高效、快速地全局寻优。

求解离散域优化问题的 ACAGA 算法的伪代码如下：

For *iteration* = 0 to Generation do

对参加进化的每个个体的变量 α，β，ρ，q_0 进行随机编码；

从个体中随机地选择 4 个；

根据给出的 4 个变量的值，求适应度函数值，即 4 个个体分别进行 TSP 寻径；

第 1 步：初始化

$t=0$；//t 是计时器

$N_c=0$；//N_c 是循环计数器

对所有的边(i,j)上的信息素赋初始值 $\tau_{ij}(t)=const$，$\Delta\tau_{ij}=0$；

将 4 只蚂蚁随机地放置在 n 个节点上；

第 2 步：$s=1$；//变量 s 是蚂蚁 k 在一次寻径过程中走的步数

For $k=1$ to 4 do

将第 k 只蚂蚁的初始节点放入数组 $tabu_k(s)$；

$J_k(s)=\{1, 2, 3, \cdots, n\}-tabu_k(s)$；//$J_k(s)$是蚂蚁 k 在第 s 步时尚未经过的节点集合

第 3 步：$s=s+1$；

For $k=1$ to 4 do

根据状态转移概率公式选择蚂蚁 k 的下一跳节点 j；//蚂蚁 k 在 t 时刻处在节点 $i=tabu_k(s-1)$

将蚂蚁 k 放置于节点 j，并将节点 j 插入数组 $tabu_k(s)$；

$tabu_k(s)=j$；$J_k(s)=J_k(s-1)-j$；

对链路进行局部信息素更新；

重复第 3 步，直到 $s=n$

第 4 步：For $k=1$ to 4 do

将蚂蚁 k 从节点 $tabu_k(n)$ 移到 $tabu_k(1)$；//蚂蚁回到初始节点，完成一次回路寻径

计算蚂蚁 k 的路由长度 L_k，并比较其大小；

求得最短长度 L_{best} 及其对应的 k_{best} 和 $tabu_{kbest}$；

对链路进行全局信息素更新；

得到任意条边的信息素浓度值 $\tau_{ij}(t+n)$；

第5步：$t=t+n$；

$N_c=N_c+1$；

If($N_c<N_{cmax}$) and 没有停滞行为

Then

{清空所有 $tabu$ 列表；

跳转到第2步；}

else

将寻径后的最短路径长度作为适应度函数；

从被选的4个个体中选出2个最优个体；

进行交叉和变异操作生成2个子个体；

替代4个个体中最差的2个，放入待进化的个体中；

End for

输出最优结果

上述伪代码流程中，针对参加遗传运算的每只蚂蚁，对蚁群算法中的4个参数 α，β，ρ，q_0 进行了28 bit 编码，其中每一参数占用7 bit。

（2）连续域蚁群遗传算法

邵晓巍等人提出的“利用信息量留存的蚁群遗传算法”将空间进行均匀分割，基于这些子空间选取初始种群，并定义每个子空间的初始信息量，遗传操作中根据信息量的留存情况来控制个体选择。因为在求解优化问题前，通常不会有全局最优点在解空间位置分布的信息，因此希望算法的搜索种群能够均匀地分散在解空间。这样可以降低发生过早收敛的可能性；而采用蚁群算法中“信息量留存”的思想，可保证算法能够快速收敛到具有最优（次优）解的子空间。算法设计如下。

将全局优化问题定义如下：

$$\max f(x) \qquad (6.25)$$
$$\text{s.t.} \quad l\leqslant x\leqslant u$$

其中，$\boldsymbol{x}=\{x_1, x_2, \cdots, x_n\}$，变量 x_i 的定义域为 $[l_i, u_i]$；$\boldsymbol{l}=\{l_1, l_2, \cdots, l_n\}$；$\boldsymbol{u}=\{u_1, u_2, \cdots, u_n\}$；$f(x)$ 表示目标函数，n 为维数。改进后的连续域蚁群遗传算法的主要步骤如下。

① 将解空间按一定的原则分解成若干子空间

针对全局最优问题的维数和每一维定义域的大小对解空间加以分解。这里以二维优化问题为例，设 $x=\{x_1, x_2\}$，其定义域分别为 $[l_1, u_1]$ 和 $[l_2, u_2]$，定义域均匀地分解为 $M\times N$ 个子空间 E_{ij}，其中 $i=1, 2, \cdots, M$，$j=1, 2, \cdots, N$，且子区域的区间长度为

$$\begin{cases} D_{1L}=\dfrac{u_1-l_1}{M} \\ D_{2L}=\dfrac{u_2-l_2}{N} \end{cases} \tag{6.26}$$

E_{ij} 的左右边界分别为 x_{1iL}，x_{2jL} 和 x_{1iR}，x_{2jR}，即有

$$\begin{cases} x_{1iL}=l_1+(i-1)D_{1L} \\ x_{2jL}=l_2+(j-1)D_{2L} \\ x_{1iR}=l_1+iD_{1L} \\ x_{2jR}=l_2+jD_{2L} \end{cases} \tag{6.27}$$

② 确定初始种群并标定各个子空间

a. 初始种群的产生：在第（1）步中确定的每个子空间中随机产生一个个体，所有个体组成初始种群。与子空间 E_{ij} 相对应的个体定义为 $A_{ij}(1)$，括号中的 1 表示整个算法的第一代个体，则初始种群可表示为 $\{A_{ij}(1)\}$，其中 $i=1, 2, \cdots, M$，$j=1, 2, \cdots, N$，种群规模为 $M\times N$。

b. 子空间的初始标定：每个子空间 E_{ij} 都由一个信息量 Ph_{ij} 标定，各个子空间信息量的初始值由其中产生的初始个体的适应度值确定。当 $f(A_{ij}(1))>0$ 时，定义 $Ph_{ij}(1)=C_1 f(A_{ij}(1))$，其中 C_1 为根据问题而设定的正常数；当 $f(A_{ij}(1))<0$ 时，定义 $Ph_{ij}(1)=\dfrac{C_3}{C_2+f(A_{ij}(1))}$，$f(A_{ij})$ 为个体 A_{ij} 的适应度值。C_2 和 C_3 的设定同 C_1。

③ 对种群进行蚁群遗传操作

a. 选择操作：选择操作的主要目的是为了避免基因缺失，以提高全局收敛性和计算效率。蚁群遗传算法选择操作的基本思想是：第 k 代中的个体 l 被选中的概率是其个体适应度值和所处子空间信息量的函数。群体规模为 $M\times N$，第 k 代中个体 l 的适应度值为 $f(A_l(k))$，个体 l 所处子空间的上一代中标定的信息量为 $Ph_l(k-1)$，则个体 l 被选中的概率为

$$P_l(k)=\frac{Ph_l^{\alpha}(k-l)f^{\beta}(A_l(k))}{\sum\limits_{i=1}^{M\times N} Ph_i^{\alpha}(k-l)f^{\beta}(A_i(k))} \tag{6.28}$$

b. 交叉操作：交叉操作是蚁群遗传算法中产生新个体的主要方法，可以针对具体问题，根据编码方法的不同选择各种常用的交叉操作和交叉概率。

c. 变异操作：变异操作是产生新个体的辅助方法，同时也决定了蚁群遗传算法的局部搜索能力。与交叉操作相似，可根据具体问题选取具体的变异方法和变异概率。

d. 子空间信息素的更新：随着种群一代代进化，各子空间的信息量也不断积累，在积累过程中必须对残留的信息量按照公式（6.29）进行更新处理，并根据第（2）步确定第一代中各子空间的信息量。

$$Ph_{ij}(k)=(1-\rho)Ph_{ij}(k-1)+Ph'_{ij}(k) \tag{6.29}$$

其中，$Ph_{ij}(k)$表示第 k 代子空间 ij 上的信息量；$Ph_{ij}(k-1)$表示第 $k-1$ 代子空间 ij 上的信息量；$Ph'_{ij}(k)$表示第 k 代遗传操作后子空间 ij 上加入的信息量。不妨设在子空间 ij 上，第 k 代以前具有最大适应度值的个体为 A_{ijmax}，第 k 代选择、交叉和变异操作后，具有最大适应度值的个体为 $A_{ijmax}(k)$。如果 $f(A_{ijmax}(k))>f(A_{ijmax})$，则 $A_{ijmax}=A_{ijmax}(k)$；否则，A_{ijmax} 不变。当 $f(A_{ijmax})>0$ 时，定义

$$Ph'_{ij}(k)=C_1f(A_{ijmax}) \tag{6.30}$$

当 $f(A_{ijmax})<0$ 时，定义

$$Ph'_{ij}(k)=\frac{C_3}{C_2-f(A_{ijmax})} \tag{6.31}$$

其中，C_1、C_2、C_3 为根据问题而设定的正常数，且 $C_2>C_3$；$f(A_{ijmax})$为个体 A_{ijmax}的适应度值。如果 k 代中某一子空间没有个体，则 A_{ijmax}保持不变。

子空间 ij 中具有最大适应度值的个体 A_{ijmax}是随着蚁群遗传操作一代代保留下来的，所以不必每一代都比较 ij 中所有个体的适应度值，只需同该代中交叉和变异产生的新个体的适应度值进行比较即可。结合各子空间内的初始信息量值，利用公式（6.29）可保证各子空间内的信息量随遗传进化而不断积累和更新。

④ 结束

当群体中的最优个体满足一定要求或总代数达到一定数量时，结束进化操作。

6.5 蚁群算法的典型应用

以蚁群算法为代表的群智能已成为当今分布式人工智能研究的一个

热点，并越来越多地被应用于企业的运转模式、生产计划制定和物流管理的研究。从美国五角大楼的“群体战略”（Swarm Strategy），到英国电信公司的基于电子蚂蚁的电信网络管理试验；从英国联合利华公司的基于蚁群算法的生产计划管理软件，到美国太平洋西南航空公司基于蚁群算法的运输管理软件等，很多政府和国际著名公司纷纷采用蚁群算法等群体智能技术来改善其运转机能。

当然以上提到的这些应用只是蚁群算法应用方面的冰山一角。实际中，蚁群算法在诸多的领域都有不俗的表现，如：旅行商问题（TSP）、指派问题（QAP）、车辆路径问题（VRP）、车间作业调度问题（JSP）、电力系统、控制参数优化、参数辨识、聚类分析、故障诊断、数据挖掘、网络路由问题、机器人领域、图像处理、航迹规划、空战决策、岩土工程、化学工业、生命科学、布局优化等。由于篇幅所限，不能详尽介绍，仅介绍车辆路径问题和车间作业调度问题，着重强调蚁群算法与实际问题的切入点及算法的改进策略。

6.5.1 车辆路径问题

车辆路径问题（Vehicle Routing Problem，VRP）：也称“车辆计划”。即已知 n 个客户的位置坐标和货物需求，在可供使用车辆数量及运载能力条件的约束下，每辆车都从起点出发，完成若干客户点的运送任务后再回到起点，要求以最少的车辆数、最小的车辆总行程完成货物的派送任务。

从本质上说，TSP 问题是 VRP 问题的基本问题，而 VRP 的复杂度远高于 TSP，应用蚁群算法求解 VRP 与 TSP 主要有以下 3 个不同之处。

（1）子路径构造过程的区别

在 TSP 中，每只蚂蚁要经过所有节点；在 VRP 中，每只蚂蚁不需要经过所有节点。因此，在求解 VRP 的每次迭代中，每只蚂蚁移动次数是不确定的，只能将是否已回到原点作为路径构造完成的标志。

（2）$allowed_k$的区别

$allowed_k$的确定是蚂蚁构造路径过程中一个非常关键的问题。在 TSP 中，蚂蚁转移时只需考虑路径距离和信息量；在 VRP 中，蚂蚁转移时不但要考虑上述因素，还需考虑车辆容量的限制。

（3）可行解结构的区别

在 TSP 中，每只蚂蚁所构造出来的路径均是一个可行解；在 VRP 中，每只蚂蚁所构造的回路仅是可行解的“零部件”，各蚂蚁所构造的回路可能能够组合成一些可行解，也可能一个可行解都得不到。因此，

研究如何设计算法来尽量避免无可行解现象的出现以及如何获取可行解是蚁群算法在 VRP 领域中应用的关键。

下面介绍基于改进的最大－最小蚂蚁系统（MMAS）的有时间窗车辆路径问题（Vehicle Routing Problem with Time Window，VRPTW）。

有时间窗车辆路径问题（Vehicle Routing Problem with Time Window，VRPTW）是一个 NP 完全问题，许多学者曾试图进行求解。万旭等人应用改进的最大－最小蚁群算法（Max－Min Ant System，MMAS）求解 VRPTW，利用其并行性与分布性，进行大规模启发式搜索。

VRPTW 可以概述如下：有 n 个货物需求点（或称顾客），已知每个需求点的需求量及位置，用多辆汽车从中心仓库（或配送中心）到达这批需求点。要求必须在它的时间窗口内为每个客户服务，如果汽车提前到了客户所在地，也必须等待，直到允许为该客户服务为止。每辆汽车载重量一定，每条路线不得超过汽车载重量。每个需求点的需求必须且只能由一辆汽车来提供，目标是最小化总的汽车行驶距离和所需的汽车数目。文中将最小化汽车总的行驶路程作为第一目标，最小化车辆数目作为第二目标。

1. 算法设计

MMAS 与 AS 的主要区别在于通过将每条轨迹上的信息素限制在 $[\tau_{min}, \tau_{max}]$ 之间，较好地避免了搜索面的局部停滞（即早熟现象）。显而易见，τ_{max}与 τ_{min}的设定至关重要。因为每次迭代路径上增加的最大信息素为$\frac{1}{L(s^{gb})}$，其中 $L(s^{gb})$为全局最好解路径的长度，所以每当更新最好解时，需要同时更新 τ_{max}与 τ_{min}。τ_{max}与信息素的挥发率 $(1-\rho)$ 以及 $L(s^{gb})$成反比，而与精英蚂蚁的数量成正比。因此可按以下策略动态地确定 $\tau_{max}(t)$与 $\tau_{min}(t)$。

① 在最初信息素还未得到更新时（即产生第一代解前），采用式（6.32）和式（6.33）确定 $\tau_{max}(t)$与 $\tau_{min}(t)$。

$$\tau_{max}(t)=\frac{1}{2(1-\rho)}\cdot\frac{1}{L(s^{gb})} \tag{6.32}$$

$$\tau_{min}(t)=\frac{\tau_{max}(t)}{20} \tag{6.33}$$

② 一旦信息素更新之后，采用式（6.34）确定 $\tau_{max}(t)$

$$\tau_{max}(t)=\frac{1}{2(1-\rho)}\cdot\frac{1}{L(s^{gb})}+\frac{\sigma}{L(s^{gb})} \tag{6.34}$$

式中，σ 为精英蚂蚁的个数。$\tau_{min}(t)$的确定与式（6.33）同。

（1）最大－最小蚂蚁算法中路径的构造

每只蚂蚁选择下一个城市时，在满足车辆容量和时间窗约束的前提下，需要考虑两个方面的因素。

①通往下一个城市的路径长度以及路径上信息素的多少。

②时间窗因素的择优性，由下一个客户 j 的时间窗宽度和所在客户 i 到达下一个客户 j 的时间等因素决定。这种择优性的优先原则为：需等待时间较短优先原则和时间窗较小优先原则。

综合以上两方面的因素，第 k 条路径上的蚂蚁在城市 v_i 选择城市 v_j 的概率由式（6.35）决定

$$P_{ij}^k=\begin{cases}\bar{\omega}_1\dfrac{(\tau_{ij})^\alpha(\eta_{ij})^\beta}{\sum\limits_{h\in\Omega}(\tau_{ij})^\alpha(\eta_{ij})^\beta}+\bar{\omega}_2\dfrac{\dfrac{1}{|t_{ij}-a_j|+|t_{ij}-b_j|}}{\sum\limits_{h\in\Omega}\dfrac{1}{|t_{ih}-a_h|+|t_{ih}-b_h|}},v_j\in\Omega\\0,\qquad\qquad\text{其他}\end{cases}\tag{6.35}$$

其中，$\Omega=\{v_j\mid v_j$ 为可被访问的城市$\}\cup\{v_0\}$，v_0 为配送中心（depot）；$\bar{\omega}_1$ 和 $\bar{\omega}_2$ 为权重系数，满足 $0\leqslant\bar{\omega}_1,\ \bar{\omega}_2\leqslant1$，且 $\bar{\omega}_1+\bar{\omega}_2=1$；$[a_j,\ b_j]$为客户 j 的时间窗；t_{ij}为由客户 i 到达客户 j 的时间（即开始为客户 i 服务的时刻＋客户 i 所需的服务时间＋从客户 i 到 j 的路程消耗时间）；α 和 β 为轨迹上的信息素与该轨迹可见性的权重系数；τ_{ij}为 v_i 与 v_j 之间轨迹上的信息素；η_{ij}为轨迹的可见性，这里取 $\eta_{ij}=\dfrac{1}{d_{ij}}$，$d_{ij}$为客户 i 与客户 j 之间的距离。

（2）信息素的更新

信息素的更新有局部更新和全局更新两种方式。这里采用全局更新的方式，只将最好的蚂蚁用于信息素的更新，就是将在路径构造中排名前几位的精英蚂蚁用于信息素更新。更新规则如下：

$$\tau_{ij}^{\text{new}}=(1-\rho)\tau_{ij}^{\text{old}}+\sum_{u=1}^{\sigma-1}\Delta\tau_{ij}^u+\sigma\Delta\tau_{ij}^*\tag{6.36}$$

仅当轨迹$(v_i,\ v_j)$被排名第 μ 好的蚂蚁利用时，其信息素才增加 $\Delta\tau_{ij}^u$。$\Delta\tau_{ij}^u=\dfrac{\sigma-\mu}{L_u}$，$L_u$ 为排名为 μ 的路径长度。所有属于当前最好路径的轨迹上的信息素都被加强 $\sigma\Delta\tau_{ij}^*$，其中 $\sigma\Delta\tau_{ij}^*=\dfrac{1}{L^*}$，$L^*$ 为当前最好解路径的长度。

这种信息素更新方式收敛速度较慢，而且其全局优化性能也不明

显。为了加快其收敛速度，同时又不影响全局优化能力，在较短的时间内找到最优解，这里提出一种改进的信息素更新方法。该方法保留全局的最好解，但为了扩大信息素更新的范围，精英蚂蚁取每次迭代结果中的前几位，这时的信息素更新仍然采用式（6.36）。在迭代过程中，对出现优于上代解的本代解给予激励，对劣于上代解的本代解给予惩罚，从而加快了其收敛速度。对更新过的路径具体激励与惩罚措施如下式：

$$\tau_{ij}^{new*}=\tau_{ij}^{new}+\tau_{ij}^{new}\times\frac{L_{new}-L_{old}}{L_{old}} \tag{6.37}$$

（3）局部优化

在蚁群算法中混入局部优化算法，对每代构造的解进行改进，可以进一步缩短解路线的长度，从而加快蚁群算法的收敛速度。这里只对每代最好解进行局部优化，局部优化作用时间为所有的蚂蚁已经构造完解，但信息素还未更新之前。这里采用2－opt的局部优化方法。

（4）初始解的构造

在系统初始化时要确定 τ_{min}，τ_{max} 的值。确定它们的值首先要明确 $L(s^{gb})$ 的值，所以在最初就必须有一个较好的可行解。由于VRPTW的复杂性，产生一个可行的初始解不是一件容易的事情。这里采用一种基于客户时间窗下限较小优先的快速产生初始解的算法，其时间复杂度为 $O(n^2\log_2 n)$，明显快于节约法 $O(n^4)$。该算法选择下一个客户的策略为，从当前路径的最后一个客户出发，到所有未访问过的客户中开始服务时间最早的那个客户。只有当开始服务时间超过这个客户的时间窗时，才需要新开辟一条路径，并从未访问过的客户中重新选择出发点，将所有未访问过的客户中开始服务时间最早的那个客户作为新路径的第一个客户。算法描述如下：

第1步：初始化。

第2步：将所有未被访问过的客户放入集合 C 中。

第3步：$C=\{c_0, \cdots, c_i, \cdots, c_j, \cdots, c_{|c|-1}\}$；$C$ 中的元素按如下规则有序排列：对任意的 $i\leqslant j$，满足 W（c_i，当前路径 R 的最后一个客户）$\leqslant W$(c_j，当前路径 R 的最后一个客户)；令 $k=1$。

第4步：如果集合 C 为空，转第7步。

第5步：如果 $W(c_k, c_m)=T$，保存当前路径 R；重新开辟一条路径，从当前未被访问的客户中随机选择一个客户 c_r 作为新路径的出发点，$C=C-c_r$；转第3步。

第6步：如果客户 c_k 为当前路径 R 的合法客户，将 c_k 加入当前路径 R 当中；$C=C-c_k$；$k=k+1$；转第4步。

第 7 步：输出计算结果，程序结束。

其中，$W(c_i, c_k)$的返回值为 max(c_m 的开始服务时间 + c_m 的服务所需时间 + 从 c_m 到 c_i 的行驶时间 a_{ci})。若到达 c_i 的时间晚于 b_{ci}，则 $W(c_i, c_m)$返回一个很大的 T 值，表示该客户不能加入路径。

2. 算例分析

将以上改进的 MMAS 算法应用于 Marius 在文献中所述的基准 VRPTW 实例，这些实例共 56 个。这 56 个实例中的每个例子都含有 100 个客户和一个中心仓库，并规定了车辆负载、客户的时间窗和车辆运行时间。实例分为 6 大类，其中 R1 和 R2 中的客户为随机分布，C1 和 C2 中的客户有聚集趋势，RC1 和 RC2 的客户兼有 R 类和 C 类的特征。

实验开始，在每个城市都放置一只蚂蚁，蚂蚁的个数与客户的数目相同。设置 $\alpha = 1.0$，$\beta = 5.0$；信息素挥发系数 ρ 取值在 0.02 到 0.3 之间；σ 的取值一般在 3 ~ 6 之间较合适；$\bar{\omega}_1$ 和 $\bar{\omega}_2$ 的设定需根据具体情况而定，一般来说，$0.5 < \bar{\omega}_1 < 1.0$，$0 < \bar{\omega}_2 < 0.5$，路径上信息素初始化为 τ_{max}。

在 MMAS 中，一般将路径上的信息素初始化为上限值 τ_{max}，这样可使系统具有更好的全局搜索能力。为了说明这样做的好处，这里将分别初始化为 τ_{max}和 τ_{min}的结果做了对比，如图 6.7 所示。图 6.7 分别描述了这两种情况下对实例 RC101 所做实验中迭代次数与获得第一目标最好解的关系，可以看出初始化为上限值虽收敛较慢，但可获得更好的解。其中 DV 表示偏离最优目标已知最优解的百分比，N 表示迭代次数。表 6.3 中比较了原来的 MMAS 与改进后的 MMAS。很明显，采用改进后的信息素更新方式会有更快的收敛速度和更好的结果。

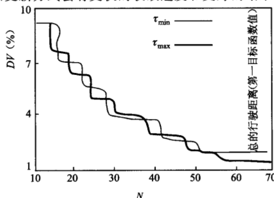


图 6.7 信息素初始化为 τ_{min}和 τ_{max}的结果对比

图6.8描述了在实例RC102中获得的第一目标值与CPU运行时间的关系。从图中可以看出，算法在运行初期就能使当前解得到很好的改善，并且收敛速度较快。实验结果与当前已知最优解的比较见表6.4，改进后MMAS的平均值为对某一实例进行10次实验所得解的平均。表6.4中给出的解分为两部分，“/”之前为第一目标值，“/”之后为第二目标值。

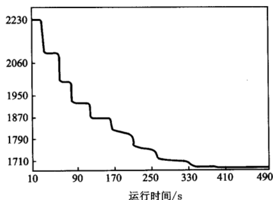


图6.8　获得的第一目标值与CPU运行时间之间的关系

表6.3　MMAS获得的最好结果与历经的迭代次数

实例	未改进的MMAS		改进后的MMAS	
	DV(%)	N	DV(%)	N
RC201	2.4	64	0.7	51
RC203	2.3	57	1.3	48
RC205	2.3	69	1.1	63
RC207	2.6	55	1.0	53
C202	2.2	54	1.3	54
C204	1.9	67	1.0	62
C206	2.5	68	1.4	64
C208	2.3	58	0.9	57

表 6.4 改进后 MMAS 的平均值及实际最优值与已知最优值的比较

实例	R101	R102	R103	R104	R105	R106
已知	1650.80/19	1486.12/17	1292.68/13	1007.31/9	1377.11/14	1252.03/12
平均	1701.58/18	1554.91/17	1338.57/13	1050.72/9	1412.14/13	1303.61/12
实际	1660.71/18	1503.47/17	1307.74/13	1032.29/9	1374.03/13	1266.74/12
实例	R107	R108	R109	R110	R111	R112
已知	1104.66/10	963.99/9	1194.73/11	1124.40/10	1096.72/10	982.14/9
平均	1148.15/10	1002.17/9	1247.32/11	1161.25/10	1133.94/10	1025.88/9
实际	1121.94/10	979.64/9	1217.09/11	1155.43/10	1120.57/10	1004.71/9

由表 6.4 可见，改进的 MMAS 的最好解已经很接近最优解，个别实例（如实例 R101 和实例 R105）的第二目标值还优于当前最好解，说明采用 MMAS 解决 VRPTW 是非常有效的。该系统有如下优点：

①改进的 MMAS 可以方便地解决其他车辆路径问题，若没有时间窗的 VRP，只需将 $\bar{\omega}_2$ 设为 0 即可。

②改进的 MMAS 可被用于多供货点 VRP 问题（VRP with Multi - Depot）。

③通过实时设定某条路径上信息素的上下限，可以将其用于动态实时的 VRP。

6.5.2 车间作业调度问题

车间作业调度（Job - shop Scheduling Problem，JSP）主要是针对一项可分解的工作（如生产制造），探讨在尽可能满足约束条件（如交货期、工艺路线、资源状况）的前提下，通过下达生产指令，安排其组成部分（操作）使用哪些资源、其加工时间及加工的先后顺序，以获得产品制造时间或成本的优化。JSP 问题是典型的 NP - hard 问题，目前已成为 CIMS 领域内的重要研究课题。JSP 问题的特征模型可描述如下：

①存在 j 个工作（job）和 m 台机器（Machine）。

②每个工作由一系列操作（或者任务/Task/Operation）组成。

③操作的执行次序遵循严格的串行顺序。

④在特定时间，每个操作需要一台特定机器完成。

⑤每台机器在同一时刻不能同时完成不同的工作。

⑥同一时刻，同一工作的各个操作不能并发执行。

⑦问题是如何求得从第一个操作开始到最后一个操作结束的最小时

间间隔（Makespan）。

1. 蚁群算法与混流装配线调度

混流装配线（Sequencing Mixed Models on Assembly Line，SMMAL）是JIT生产方式的具体应用之一，它可以在不增加产品库存的条件下满足用户的多样化需求。所谓混流装配线，是指在一定时间内，在一条生产线上生产出多种不同型号的产品，产品的品种可以随顾客需求的变化而变化。

（1）算法设计

采用丰田公司提出的调度目标函数，以汽车组装为例，即在组装所有车辆的过程中，所确定的组装顺序应使各零部件的使用速率均匀化。如果不同型号的汽车消耗零部件的种类大致相同。那么原问题可简化为单级SMMAL。则问题可描述为

$$\min \sum_{j=1}^{D} \sum_{i=1}^{n} \sum_{p=1}^{m} (k\alpha_p - b_{ip} - \beta_{j-1,p})^2 x_{ji} \tag{6.38}$$

$$x_{ji} = \begin{cases} 1, & \text{车型 } i \text{ 在调度中的 } j \text{ 位置} \\ 0, & \text{其他} \end{cases} \tag{6.39}$$

$$\alpha_p = \frac{\sum_i d_i b_{ip}}{D} \tag{6.40}$$

其中，i 表示车型数的标号；n 表示需要装配的车型数；m 表示装配线上需要的零部件种类总数；p 表示生产调度中子装配的标号；j 表示车型调度结果（即排序位置）的标号；D 表示在一个生产循环中需要组装的各种车型的总和；d_i 表示在一个生产循环中车型 i 的数量；b_{ip} 表示生产每辆 i 车型需要零部件 p 的数量；α_p 表示零部件 p 的理想使用速率；$\beta_{j-1,p}$ 表示组装线调度中前 $j-1$ 台车消耗零部件 p 的数量和，为

$$\beta_{jp} = \beta_{j-1,p} + x_{ji}\beta_{ip}，\text{其中 } \beta_{0,p} = 0 \tag{6.41}$$

调度的目标函数如式（6.38）所示，采用如图6.9所示的搜索空间定义，列表示排序阶段（用 j 表示），行表示每个阶段可供选择的车型（用 i 表示），圆圈的大小表示选择概率的大小，蚁群算法就是不断改变圆圈的大小，最终寻找到满意的可行解。举例说明混流装配线排序搜索空间的例子。有3种车型A、B、C排序，每个生产循环需A车型3辆，B车型2辆，C车型1辆，则每个循环共需生产6辆车（$D=6$）。列表示6个排序阶段，行表示有3种车型可以选择。图6.9（a）所示的是搜索的初始状态，圆圈由局部搜索值和激素综合而成，图6.9（b）表示经过若干次迭代之后，搜索空间变化。这时候，最可能的可行解是B-A-C-A-B-A。利用这种表示方法可以降低问题描述的维数，该

问题规模为 $O(n \cdot m)$，其中 n 是车型数，m 是排序长度。

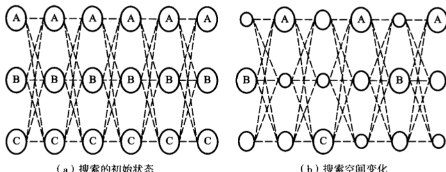


图 6.9 简单 SMMAL 排序的搜索空间举例

①局部搜索的计算

$$\eta_{ij} = \frac{Q}{\sum_{k=1}^{D} \sum_{i=1}^{n} \sum_{p=1}^{m} (k\alpha_p - b_{ip} - \beta_{j-1,p})^2 x_{ki}} \tag{6.42}$$

其中，k 表示第 k 只蚂蚁；Q 是一个常数，调节 η_{ij} 的大小。局部搜索 η_{ij} 采用贪婪法（与目标追随法一致）。目标追随法的思路是，每一步均从当前可选择策略中选取，使目标函数值增加最少的策略，即在确定第 $n+1$ 台车辆的车型时，如有多种车型可供选择，则从中选择一种车型，使第 $n+1$ 台车辆组装时各零部件的使用速率最均匀。若每步只考虑当前的状态，而不考虑全局状态，这样得到的结果常常为局部最优解。但是蚁群算法为每一个可选择的车型 i 计算 η_{ij}，最后再结合信息素的作用，再做出车型的选择。

②状态转移概率

状态转移概率公式

$$p_{ij}^k(t) = \begin{cases} \dfrac{\alpha\tau_{ij} + (1-\alpha)\eta_{ij}}{\sum_{j \notin tabu_k} (\alpha\tau_{ij} + (1-\alpha)\eta_{ij})}, (i) \notin tabu_k \\ 0, \quad 其他 \end{cases} \tag{6.43}$$

其中，α 表示信息素的相对重要性；$\tau_{ij}(t)$ 在 t 时刻，i 车型放置在 j 位次上信息激素的数量；η_{ij} 是当 $j-1$ 个车型顺序排定后，将 i 车型放置在 j 位次上目标函数的值；$tabu_k(t)$ 表示存放在 t 时刻，第 k 只蚂蚁不可以走的节点；$p_{ij}^k(t)$ 表示在 t 时刻，第 k 只蚂蚁选择将 i 车型放置在 j 位次上的概率。

③信息素更新规则

$$\Delta\tau_{ij}^{k}=\begin{cases}\tau_{0}\left(1-\dfrac{Z_{\mathrm{cutr}}-LB}{\bar{Z}-LB}\right), & \text{车型 } i \text{ 在调度中的 } j \text{ 位置}\\ 0, & \text{其他}\end{cases} \tag{6.44}$$

其中，$\Delta\tau_{ij}^{k}$表示在 t 时刻，第 k 只蚂蚁在 i 车型 j 位次上放置的信息激素；LB 表示目标函数的下限值，$\bar{Z}$ 表示目前目标函数的平均值，Z_{cutr}是当前的目标函数值。这种动态标记的方法可以在搜索的过程中加大可行解间信息激素的差别，避免过早的收敛，如图 6.10 所示。

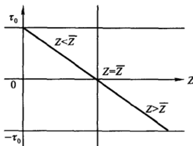


图 6.10 线性动态标注

而 $\Delta\tau_{ij}=\sum_{k=1}^{\mathrm{n-ant}}\Delta\tau_{ij}^{k}$。因此可将信息素更新策略表示为

$$\tau_{ij}(t+n)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij} \tag{6.45}$$

式中，ρ 表示信息激素的消散速率。

（2）算例分析

这里采用文献［30］中的算例数据，见表 6.5。经过多种组合试验，得到蚁群算法最优组合参数为 $\rho=0.9$，$\alpha=0.2$，$Q=20\ 000$，$N_{\mathrm{cmax}}=400$，$n=5$，目标函数值 2 859.8，排序结果为 C－A－D－E－B－A－D－E－A－C－A－B－E－D－A－C。表 6.6 列出了文中的蚁群算法与文献中给出的目标追随法、遗传算法和模拟退火算法的比较结果，蚁群算法的求解性能优于其他启发式算法。

表 6.5 各种车型的物料单和需求的子装配数

车型	每个生产循环生产的各种车型数	子装配									
		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
A	5	0	17	9	0	4	0	0	18	0	0
B	2	12	13	0	11	0	0	0	1	17	17
C	3	2	4	0	19	0	12	6	4	9	3

续表

车型	每个生产循环生产的各种车型数	子装配									
		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
D	3	0	15	0	19	0	15	9	9	0	6
E	3	0	0	5	7	8	10	4	0	0	0
每个生产循环需求的子装配数		30	168	60	157	44	111	57	131	61	61

表 6.6 蚁群算法与其他启发式算法的对比

算法名称	目标函数	结果改善的百分比（%）
目标追随法	3293	13
遗传算法	3073	6
模拟退火算法	3162	10
蚁群算法	2859.8	-

2. 双向收敛蚁群算法与车间作业调度

为了合理高效地调度资源，解决组合优化问题，在 Job - Shop 问题图形化定义的基础上，借鉴精英策略的思路，提出使用多种挥发方式的双向收敛蚁群算法，提高了算法的效率和可用性。

为了便于比较双向收敛蚁群算法与基本蚁群算法求解 JSP 的性能，采用 Muth 和 Thompson 在 1963 年提出的 Job - Shop 6 ×6 基准问题，见表 6.7（表中 t 表示任务所需时间）。

表 6.7 Muth & Thompson 6 ×6 基准问题

项目	m, t	m, t	m, t	m, t	m, t	m, t
Job_1	3，1	1，3	2，6	4，7	6，3	5，6
Job_2	2，8	3，5	5，10	6，10	1，10	4，4
Job_3	3，5	4，4	6，8	1，9	2，1	5，7
Job_4	2，5	1，5	3，5	4，3	5，8	6，9
Job_5	3，9	2，3	5，5	6，4	1，3	4，1
Job_6	2，3	4，3	6，9	1，10	5，4	3，1

表6.7中，Job_1 的第一个任务 $Task_1$ 需在 $Machine_3$ 上完成，历时1个时间单位；第二个任务 $Task_2$ 在 $Machine_1$ 上完成，历时3个时间单位，以此类推。Job中的后续任务不能在前面任务完成之前启动。求解Job－Shop问题，就是针对每一个Machine，调度其上的任务次序。上述问题使用矩阵表示如式（6.46）～式（6.47）。

矩阵 $\boldsymbol{T}$ 表示每个工作的任务调度顺序，矩阵 $\boldsymbol{P}$ 表示相应的时间间隔。采用文献［7］中的定义方法，将以上问题转换成如图6.11所示的结构。

$$\boldsymbol{T}=\begin{bmatrix} m_3 & m_1 & m_2 & m_4 & m_6 & m_5 \\ m_2 & m_3 & m_5 & m_6 & m_1 & m_4 \\ m_3 & m_4 & m_6 & m_1 & m_2 & m_5 \\ m_2 & m_1 & m_3 & m_4 & m_5 & m_6 \\ m_3 & m_2 & m_5 & m_6 & m_1 & m_4 \\ m_2 & m_4 & m_6 & m_1 & m_5 & m_3 \end{bmatrix} \tag{6.46}$$

$$\boldsymbol{P}=\begin{bmatrix} t_1 & t_3 & t_6 & t_7 & t_3 & t_6 \\ t_8 & t_5 & t_{10} & t_{10} & t_{10} & t_4 \\ t_5 & t_4 & t_8 & t_9 & t_1 & t_7 \\ t_5 & t_5 & t_5 & t_3 & t_8 & t_9 \\ t_9 & t_3 & t_5 & t_4 & t_3 & t_1 \\ t_3 & t_3 & t_9 & t_{10} & t_4 & t_1 \end{bmatrix} \tag{6.47}$$

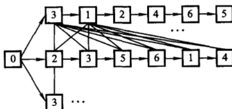


图6.11 JSP的图形化定义

图6.11所示的结构由37个节点组成，增加了一个虚拟起点0，从点0开始可以提供通往 Job_1，Job_2，…，Job_6 的单向通路，此外的节点 $P_{i,j}(1\leqslant i,\ j\leqslant 6)$ 表示Machine矩阵 $\boldsymbol{T}$ 中相应位置的点，即 T_{ij} 中的Machine代号。因此，图6.11中的第 i 行（除起始点0以外）表示表6.7中的第 i 个Job。每个工作内的各个任务由有向弧连接，各个工作之间的任务由无向弧连接。

对图 6.11 所示的图结构作如下定义。

定义 6.4（操作 Operation）：操作用节点表示，定义操作为

Operation = 〈sTime, eTime, isDone, NEXT〉

其中，sTime 和 eTime 分别是本操作的开始和结束时间；isDone 表示本节点代表的操作是否已经完成，或者蚂蚁是否已经走过节点；NEXT 表示下一步的结构，包含了从本节点能够到达的节点表和到达这些节点的路径上信息素的数量表。

定义 6.5（图 Graph）：图为操作的集合

Graph =〈Operation, Machine〉

其中，Machine 为表 6.7 中表示操作所占用的机器的矩阵。在 Operation 节点中已经包含节点之间的连接，因此不需要定义节点之间的关系。

定义 6.6（虚拟起始点 Start Point:）

StartPoint =〈NEXT, nextMachine〉

虚拟起始点只连接每个工作的第一个操作。操作所占用的时间从 NEXT 结构中的 Operation 体现，因此虚拟起始点不占用操作时间。

（1）算法设计

双向收敛蚁群算法，将历史最差解看作目前不可接受的解，对其进行惩罚，可以引导其他蚂蚁尽量远离历史最差解，放弃将最差解的组成部分组合成其他解的机会，从而加速算法的收敛。下面具体介绍该算法。

①生成一代蚂蚁

根据算法规定的数量放出蚂蚁，为保证操作符合工作的调度顺序，在每只蚂蚁寻找路径的过程中，首先判断目的节点的前驱是否已经完成，在前驱已经完成并且本身尚未完成的所有节点中，使用信息素的浓度作为概率选取下一步目标。在目标的选取过程中，借鉴遗传算法中常见的轮盘方法（Roulette Wheel）决定。为保证蚂蚁遍历的次序符合 Job 操作的次序要求，使用下列原则：

a. Job 中同一行的节点完成后不能直接转向自己的非直接后继点。

b. 使用评价函数计算经过的路径代表的时间间隔 Makespan 时，遵守任务的先后次序，使蚂蚁行走时路径不代表任务次序。

c. 次序的含义在计算 Makespan 时被加到路径上。

在双向收敛的蚁群算法中，蚂蚁无需每走出一步都对下一步的机器占用和工序次序做判断，大大降低了计算量，提高了算法的效率。同时，算法在计算状态转移概率的过程中，不按照传统的方法将路径长度考虑在内。假定蚂蚁行走的过程中不会重复已经走过的路径，蚂蚁选择下一条可能路径的状态转换规则是

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{j \in allowednodes} \tau_{ij}(t)} \tag{6.48}$$

②评价和激励

当本代蚁群中的所有蚂蚁完成对有向图中所有点的遍历后，使用评价函数对得到的所有路径进行评价，并按以下规则更新信息素：

$$\tau_{ij}(t+\Delta t) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t+\Delta t) \tag{6.49}$$

其中，式（6.49）的第一部分表示每一代蚂蚁走完全程后所有信息素的挥发，初始状态的信息素随机给出；第二部分表示信息素的修改，计算公式为

$$\Delta\tau_{ij}(t+\Delta t) = \begin{cases} \dfrac{Q}{f(\text{bestRoad})}, & t+n\text{ 过程中的最优值} \\ -\dfrac{Q'}{f(\text{bestRoad})}, & t+n\text{ 过程中的最差值} \\ 0, & \text{其他情况} \end{cases} \tag{6.50}$$

其中，Q 表示单位路径上的信息量；Q'表示单位路径上的用于惩罚最差值的信息量；$f(\,)$表示路径评价函数。

基本 ACO 使用参数调节的方法避免算法陷入局部最优，这种方法取决于具体的参数数值，往往导致一套参数对应于一个具体问题，降低了算法的通用性。而双向收敛蚁群算法从两个方向进行反馈，很大程度上避免了对参数的依赖，同时加快了算法的收敛。

③循环执行

如果已经收敛于最优值或者到达最大蚂蚁代数，退出循环并输出算法结果；否则，跳转步骤①。

④评价函数

评价函数的基本思路是：规定每只蚂蚁经过的路径中，前面节点的开始时间不会落在后面节点之后。由于蚂蚁在寻径过程中已经考虑到了操作在工作中的先后次序，结果中的节点串将表示不同工作的时间次序。具体算法如下：

```
求时间间隔 Makespan
//初始化第一个操作的时间
O1. startTime = 0;
O1. endTime = O1. executeTime;
MO1. Time = O1. endTime;
Makespan = 0;
For(int i = 2; i < = m · n; i + + )//根据蚂蚁的路径找到相应操作
```

的起止时间

```
{
  O_i.startTime = O_{i-1}.endTime; //本次操作只能在前面操作已经完成的情况下执行
  找到 O_i 对应的机器 M_{Oi};
  if(M_{Oi}.Time > O_i.startTime) then//如果这时机器被占用
  O_i.startTime = M_{Oi}.Time; //等机器释放后开始
  O_i.endTime = O_i.startTime + O_i.executeTime;
  M_{Oi}.Time = O_i.endTime;
}
for(int j = 1; j < = NumberOfMachine; j + +)//求出完成所有工作所需时间
{
  if Makespan < M_j.Time then Makespan = M_j.Time;
}
```

使用双向收敛蚁群算法可以求出每只蚂蚁遍历图的所有节点之后得出的 Job－Shop 的解。根据这个解，对所有蚂蚁路径中最优的一条和最差的一条使用式（6.50）改变信息素。

（2）算例分析

这里采用 Muth 和 Thompson 的 6×6 基准问题作为仿真算例。表 6.8 比较了双向收敛蚁群算法和基本蚁群算法所得到的解，所得结果为 5 次求解后的平均值。

表 6.8 双向收敛蚁群算法和基本蚁群算法最优解的比较

算法类型	蚂蚁数目	循环次数	蚂蚁数目	ρ	收敛结果
基本蚁群算法	36	3 000	108 000	0.010	55
双向收敛蚁群算法	200	311	62 200	0.017	55

由表 6.8 可见，双向收敛蚁群算法使用了较多蚂蚁进行并行搜索，提高了搜索过程的挥发系数，从而在较少的代数中得到了 JSP 的解。

问题与思考

1. 根据基本蚁群算法（AS）的程序结构流程写出基本蚁群算法的伪码表示。
2. 对基本蚁群算法进行空间复杂度分析。
3. 比较 AS（基本蚁群算法）与 ACS（蚁群算法）、MMAS（最大－最小蚁群

算法）的异同，并在计算机上实现以上三种算法。

4. 在思考题3的基础上，通过实验分析信息素 τ_{ij} 和启发式因子 η_{ij} 对蚁群算法性能的影响；并改变参数 α，β，ρ，m，Q 的大小，分析其对蚁群算法性能的影响。

5. 多旅行商问题，给定 n 个城市的集合，m 个旅行商从不同城市出发，分别走一条旅行路线，使得每个城市有且仅有一个旅行商经过，使总旅行路程最短。试为该问题设计编码方案，并按最大－最小蚁群算法（MMAS）设计求解问题的方法。要求写明状态转移概率，信息素的更新策略，并画程序框图。

6. 工作指派问题简述如下：n 个工作可以由 n 个工人分别完成。工人 i 完成工作 j 的时间为 d_{ij}。问如何安排可使总工作时间达到极小。建立数学模型，并按蚁群算法设计求解问题的算法。要求写明状态转移概率，信息素的更新策略，画出程序框图。

参考文献

[1] Bonabeau E, Dorigo M, Theraulaz G. Inspiration for optimization from social insect behavior [J]. Nature, 2000, 406 (6): 39－42.

[2] Daniel M, Martin M. Ant colony optimization with global pheromone evaluation for scheduling a single machine [J]. Applied Intelligence, 2003, 18: 105－111.

[3] Dorigo M, Caro G D, Gambardella L M. Ant algorithms for discrete optimization [J]. Artificial Life, 1999, 5 (2): 137－172.

[4] Dorigo M, Gambardella L M. A study of some properties of Ant－Q. Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, Berlin, September 22－27, 1996 [C]. Berlin: Springer Verlag, c1996, 656－665.

[5] Dorigo M, Gambardella L M. Ant colony system: a cooperative learning approach to traveling salesman problem [J]. IEEE Transactions on Evolutionary Computation, 1997, 1 (1): 53－66.

[6] Dorigo M, Luca M. The Ant－Q: algorithms applied to the nuclear reload problem [J]. Annals of Nuclear Energy, 2002, 29 (12): 1455－1470.

[7] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents [J]. IEEE Transactions on Systems, Man, and Cybernetics－Part B, 1996, 26 (1): 29－41.

[8] Dorigo M. Optimization, learning and natural algorithms [D]. Italy: Politecnico diMilano, Department of Electronics, 1992.

[9] Dréo J, Siarry P. Continuous interacting ant colony algorithm dased on dense heterarchy. Future Generation Computer Systems [J], 2004, 20 (5): 841 - 856.

[10] Gambardella L M, Dorigo M. Ant - Q: a reinforcement learning approach to the traveling salesman problem. Proceedings of the 12th International Conference on Machine Learning, Tahoe City, July 9 - 12, 1995 [C]. CA: Morgan Kaufman, c1995, 252 - 260.

[11] Gutjahr W J. A graph - based ant system and its convergence [J]. Future Generation Computer Systems, 2000, 16 (8): 873 - 888.

[12] Jackson D E, Holcombe M, Ratnieks F L W. Trail geometry gives polarity to ant foraging networks [J]. Nature, 2004, 432 (7019): 907 - 909.

[13] Jain A S, Meeran S. Deterministic job - shop scheduling: past, present and future [J]. European Journal of Operational Research, 1999, 113 (2): 390 - 434.

[14] Marius M, Solomon M. Algorithms for vehicle routing and scheduling problems with time window constraints [J]. Operations Research, 1987, 35 (2): 763 - 781.

[15] Merkle D, Middendorf M. Modeling the dynamics of ant colony optimization [J]. Evolutionary Computation, 2002, 10 (3): 235 - 262.

[16] Michael J B K, Jean - Bernard B, Laurent K. Ant - like task and recruitment in cooperative robots [J]. Nature, 2000, 406 (31): 992 - 995.

[17] Reimann M, Doerner K, Richard F H. D - Ants: saving based ants divide and conquer the vehicle routing problem [J]. Computer & Operations Research, 2004, 31 (4): 563 - 591.

[18] Stützle T, Hoos H H. MAX - MIN ant system [J]. Future Generation Computer Systems, 2000, 16 (9): 889 - 914.

[19] 程志刚. 连续蚁群优化算法的研究及其化工应用 [D]. 杭州: 浙江大学, 2005.

[20] 段海滨. 蚁群算法原理及其应用 [M]. 北京: 科学出版社, 2005.

[21] 刘志硕, 申金升, 柴跃廷. 基于自适应蚁群算法的车辆路径问题研究 [J]。控制与决策, 2005, 20 (5): 562 - 566.

[22] 邵晓巍，邵长胜，赵长安．利用信息量留存的蚁群遗传算法[J]。控制与决策，2004，19（10）：1187－1189，1193.
[23] 孙力娟，王良俊，王汝传．改进的蚁群算法及其在TSP中的应用研究[J]。通信学报，2004，25（10）：111－116.
[24] 孙新宇，万筱宁，孙林岩．蚁群算法在混流装配线调度问题中的应用[J]．信息与控制，2002，31（6）：486－490
[25] 万旭，林健良，杨晓伟．改进的最大－最小蚂蚁算法在有时间窗车辆路径问题中的应用[J]。计算机集成制造系统，2005，11（4）:572－576.
[26] 王常青，操云甫，戴国忠．用双向收敛蚁群算法解作业车间调度问题[J]．计算机集成制造系统，2004，10（7）：820－824.
[27] 徐俊刚，戴国忠，王宏安．生产调度理论和方法研究综述[J]．计算机研究与发展，2004，41（2）：257－267.
[28] 杨勇，宋晓峰，王建飞等．蚁群算法求解连续空间优化问题[J]．控制与决策，2003，18（5）：573－576.
[29] 张纪会，徐心和．一种新的进化算法—蚁群算法[J]．系统工程理论与实践，1999，19（3）：84－87.
[30] 赵伟，韩文秀，罗永泰．准时生产方式下混流装配线的调度问题[J]．管理科学学报，2000，3（4）：23－28.

第7章 粒子群优化算法

7.1 导 言

James Kennedy 和 Russell Eberhart 在 1995 年的 IEEE International Conference on Neural Networks 和 6th International Symposium on Micromachine and Human Science 上分别发表了 “Particle swarm optimization” 和 “A new optimizer using particle swarm theory” 的论文，标志着粒子群优化（Particle Swarm Optimization，PSO）算法的诞生。国内也有人译为微粒群算法。

粒子群优化由于其算法的简单，易于实现，无需梯度信息，参数少等特点在连续优化问题和离散优化问题中都表现出良好的效果，特别是因为其天然的实数编码特点适合于处理实优化问题。近年来成为国际上智能优化领域研究的热门。在算法的理论研究方面，有部分研究者对算法的收敛性进行了分析，更多的研究者致力于研究算法的结构和性能改善，包括参数分析，拓扑结构，粒子多样性保持，算法融合和性能比较等。粒子群优化算法最早应用于非线性连续函数的优化和神经元网络的训练，后来也被用于解决约束优化问题、多目标优化问题、动态优化问题等。在数据分类、数据聚类、模式识别、电信 QoS 管理、生物系统建模、流程规划、信号处理、机器人控制、决策支持以及仿真和系统辨识等方面，都表现出良好的应用前景。国内也有越来越多的学者关注粒子群优化算法的应用，将其应用于非线性规划，同步发电机辨识，车辆路径，约束布局优化，新产品组合投入，广告优化等问题。

粒子群优化算法的提出是基于对简化的社会模型的模拟。

自然界中许多生物体具有一定的群体行为，人工生命的主要研究领域之一就是探索自然界生物的群体行为，从而在计算机上构建其群体模型。通常群体行为可以由几条简单的规则进行建模，如鱼群、鸟群等。虽然每个个体具有非常简单的行为规则，但是群体行为却非常复杂。

一些科学家对鸟群或者鱼群的群体性行为进行了研究，包括计算机模拟仿真。Reynolds 和 Heppner，这两位动物学家在 1987 年和 1990 年发表的论文中都关注了鸟群群体行动中蕴涵的美学，他们发现，由数目

庞大的个体组成的鸟群飞行中可以改变方向，散开或者队形的重组等，那么一定有某种潜在的能力或者规则保证了这些同步的行为。这些科学家都认为上述行为是基于不可预知的鸟类社会行为中的群体动态学。在这些早期的模型中他们把重点都放在了个体间距的处理上，也就是让鸟群中的个体之间保持最优的距离。

1975 年，生物社会学家 Wilson E O 根据对鱼群的研究，在论文中提出："至少在理论上，鱼群的个体成员能够受益于群体中其他个体在寻找食物的过程中的发现和以前的经验，这种受益是明显的，它超过了个体之间的竞争所带来的利益消耗，不管任何时候食物资源不可预知的分散于四处"。这说明，同种生物之间信息的社会共享能够带来好处，这是 PSO 的基础。

对人类的社会行为的模拟与前者不同，最大区别在于抽象性！鸟类和鱼类是调节它们的物理运动，来避免天敌，寻找食物，优化环境的参数，比如温度等。人类调节的不仅是物理运动，还包括认知和经验变量。我们更多的是调节自己的信仰和态度，来和社会中的上流人物或者专家，或者说在某件事情上获得最优解的人保持一致。

这种不同导致了计算机仿真上的差别，至少有一个明显的因素：碰撞。两个个体即使不被绑在一块，也具有相同的态度和信仰，但是两只鸟是绝对不可能不碰撞而在空间中占据相同位置的。这是因为动物只能在三维的物理空间中运动，而人类还在抽象的多维心理空间运动，这里是碰撞自由的（Collosion - Free）。

Kennedy 和 Eberhart 对 Hepper 的模仿鸟群的模型进行了修正，以使粒子能够飞向解空间，并在最好解处降落，从而得到了粒子群优化算法。

7.2 基本原理

本节首先介绍基本粒子群优化算法，这是算法的初始版本；然后介绍粒子群优化算法的标准版本，这是目前大多数研究者所使用的版本；之后对算法的构成要素进行简单的分析；最后给出一个计算的例子。

7.2.1 基本粒子群优化算法

1. 算法原理

算法的基本原理可以描述如下。

一个由 m 个粒子（Particle）组成的群体（Swarm）在 D 维搜索空

间中以一定的速度飞行，每个粒子在搜索时，考虑到了自己搜索到的历史最好点和群体内（或邻域内）其他粒子的历史最好点，在此基础上进行位置（状态，也就是解）的变化。

第 i 个粒子的位置表示为：$\boldsymbol{x}_i=(x_{i1}, x_{i2}, \cdots, x_{iD})$

第 i 个粒子的速度表示为：$\boldsymbol{v}_i=(v_{i1}, v_{i2}, \cdots, v_{iD})$，$1\leqslant i\leqslant m$，$1\leqslant d\leqslant D$

第 i 个粒子经历过的历史最好点表示为：$\boldsymbol{p}_i=(p_{i1}, p_{i2}, \cdots, p_{iD})$

群体内（或邻域内）所有粒子所经过的最好的点表示为：$\boldsymbol{p}_g=(p_{g1}, p_{g2}, \cdots, p_{gD})$。

一般来说，粒子的位置和速度都是在连续的实数空间内进行取值。

粒子的位置和速度根据如下方程进行变化：

$$v_{iD}^{k+1}=v_{iD}^{k}+c_1\xi(p_{iD}^{k}-x_{iD}^{k})+c_2\eta(p_{gD}^{k}-x_{iD}^{k}) \tag{7.1}$$

$$x_{iD}^{k+1}=x_{iD}^{k}+v_{iD}^{k+1} \tag{7.2}$$

其中，c_1 和 c_2 称为学习因子（Learning Factor）或加速系数（Acceleration Coefficient），一般为正常数。学习因子使粒子具有自我总结和向群体中优秀个体学习的能力，从而向自己的历史最优点以及群体内或邻域内的历史最优点靠近。c_1 和 c_2 通常等于2。ξ，$\eta\in U[0, 1]$，是在 [0, 1] 区间内均匀分布的伪随机数。粒子的速度被限制在一个最大速度 $V_{\max}$ 的范围内。

当把群体内所有粒子都作为邻域成员时，得到粒子群优化算法的全局版本；当群体内部分成员组成邻域时得到粒子群优化算法的局部版本。局部版本中，一般有两种方式组成邻域，一种是索引号相邻的粒子组成邻域，另一种是位置相邻的粒子组成邻域。粒子群优化算法的邻域定义策略又可以称为粒子群的邻域拓扑结构。

2. 算法流程

基本粒子群优化算法的流程如下：

第 1 步：在初始化范围内，对粒子群进行随机初始化，包括随机位置和速度。

第 2 步：计算每个粒子的适应值。

第 3 步：对于每个粒子，将其适应值与所经历过的最好位置的适应值进行比较，如果更好，则将其作为粒子的个体历史最优值，用当前位置更新个体历史最好位置。

第 4 步：对每个粒子，将其历史最优适应值与群体内或邻域内所经历的最好位置的适应值进行比较，若更好，则将其作为当前的全局最好位置。

第5步：根据式（7.1）和（7.2）对粒子的速度和位置进行更新。

第6步：若未达到终止条件，则转第2步。

一般将终止条件设定为一个足够好的适应值或达到一个预设的最大迭代代数。

3. 粒子的社会行为分析

从粒子的速度更新方程（7.1）可以看到，基本粒子群优化算法中，粒子的速度主要由三部分构成。

（1）前次迭代中自身的速度

式（7.1）右侧第一项，这是粒子飞行中的惯性作用，是粒子能够进行飞行的基本保证。

（2）自我认知的部分

式（7.1）右侧第二项，表示粒子飞行中考虑到自身的经验，向自己曾经找到过的最好点靠近。

（3）社会经验的部分

式（7.1）右侧第三项，表示粒子飞行中考虑到社会的经验，向邻域中其他粒子学习，使粒子在飞行时向邻域内所有粒子曾经找到过的最好点靠近。

Kennedy 通过神经网络训练的实验研究了粒子飞行时的行为，在实验中将粒子的速度更新公式分别取以下几种情况：

（1）完全模型（Full model）：即按照原始公式（7.1）进行速度更新。

（2）只有自我认知（Cognition - only）：即速度更新时只考虑上述第1项和第2项。

（3）只有社会经验（Social - only）：即速度更新时只考虑上述第1项和第3项。

（4）无私（Selfless）：即速度更新时只考虑上述第1项和第3项，并且邻域不包括粒子本身。

这里考虑“无私”的情况是因为，在只有社会经验的模型中，如果粒子自身取得的历史最好解就是邻域最好解，那么粒子还是会被自身取得的历史最好解所吸引，这容易引起效果上的混淆，“无私”情形可以彻底去掉自身认知的影响。

实验结果表明：①对于所有的情形，最大速度 V_{max} 过小常常导致搜索的失败；而较大的 V_{max} 常使粒子飞过目标区域，这可能使粒子找到更好的区域，即，使粒子脱离局优。②上述速度更新模型按照达到规定误差所需的迭代次数从少到多依次为

只有社会经验 < 无私 < 完全模型 < 只有自我认知

这里神经网络训练的是 XOR 问题，这说明，对于简单问题，只有社会经验的模型可以最快达到收敛，这是因为粒子间的社会信息的共享导致进化速度加快。而只有自我认知的模型收敛最慢，这是因为不同的粒子间缺乏信息交流，没有社会信息的共享，导致找到最优解的概率变小。

但是，需要注意的是，收敛速度不是优化效果的唯一评价指标。特别是对于复杂的问题，只考虑社会经验，将导致粒子群体过早收敛，从而陷于局优；只考虑个体自身经验，将使群体很难收敛，进化速度过慢。相对而言，完全模型是较好的选择。

7.2.2 标准粒子群优化算法

为改善算法收敛性能，Shi 和 Eberhart 在 1998 年的论文中引入了惯性权重的概念，将速度更新方程修改为式（7.3）所示

$$v_{id}^{k+1} = \omega v_{id}^{k} + c_1\xi(p_{id}^{k} - x_{id}^{k}) + c_2\eta(p_{gd}^{k} - x_{id}^{k}) \tag{7.3}$$

其中，ω 称为惯性权重，其大小决定了对粒子当前速度继承的多少，合适的选择可以使粒子具有均衡的探索能力（Exploration，即广域搜索能力）和开发能力（Exploitation，即局部搜索能力）。可见，基本粒子群优化算法是惯性权重 $\omega = 1$ 的特殊情况。

分析和实验表明，设定 V_{max} 的作用可以通过惯性权重的调整来实现。现在的粒子群优化算法基本上使用 V_{max} 进行初始化，将 V_{max} 设定为每维变量的变化范围，而不必进行细致的选择与调节。

目前，对于粒子群优化算法的研究大多以带有惯性权重的粒子群优化算法为对象进行分析、扩展和修正，因此大多数文献中将带有惯性权重的粒子群优化算法称为粒子群优化算法的标准版本，或者称为标准粒子群优化算法；而将前述粒子群优化算法称为初始粒子群优化算法/基本粒子群优化算法，或者称为粒子群优化算法的初始版本。

7.2.3 算法构成要素

这里将对粒子群优化算法的构成要素进行概述。这些构成要素包括算法的相关参数：群体大小、学习因子、最大速度、惯性权重。也包括算法设计中的相关问题：邻域拓扑结构、粒子空间的初始化和停止准则。

1. 群体大小 *m*

m 是个整型参数。当 m 很小的时候，陷入局优的可能性很大。然

而，群体过大将导致计算时间大幅增加。并且当群体数目增长至一定水平时，再增长将不再有显著的作用。当 $m=1$ 的时候，PSO 算法变为基于个体搜索的技术，一旦陷入局优，将不可能跳出。当 m 很大时，PSO 的优化能力很好，可是收敛的速度将非常慢。

2. 学习因子 c_1 和 c_2

学习因子使粒子具有自我总结和向群体中优秀个体学习的能力，从而向群体内或邻域内最优点靠近。c_1 和 c_2 通常等于2，不过在文献中也有其他的取值。但是一般 c_1 等于 c_2，并且范围在0和4之间。

3. 最大速度：V_{max}

最大速度决定粒子在一次迭代中最大的移动距离。V_{max}较大，探索能力增强，但是粒子容易飞过最好解。V_{max}较小时，开发能力增强，但是容易陷入局优。有分析和实验表明，设定 V_{max}的作用可以通过惯性权重的调整来实现。所以现在的实验基本上使用 V_{max}进行初始化，将 V_{max}设定为每维变量的变化范围，而不必进行细致的选择与调节。

4. 惯性权重

智能优化方法的运行是否成功，探索能力和开发能力的平衡是非常关键的。对于粒子群优化算法来说，这两种能力的平衡就是靠惯性权重来实现。较大的惯性权重使粒子在自己原来的方向上具有更大的速度，从而在原方向上飞行更远，具有更好的探索能力；较小的惯性权重使粒子继承了较少的原方向的速度，从而飞行较近，具有更好的开发能力。通过调节惯性权重能够调节粒子群的搜索能力。

5. 领域拓扑结构

全局版本粒子群优化算法将整个群体作为粒子的邻域，速度快，不过有时会陷入局部最优；局部版本粒子群优化算法将索引号相近或者位置相近的个体作为粒子的邻域，收敛速度慢一点，不过很难陷入局部最优。显然，全局版本的粒子群优化算法可以看作局部版本粒子群优化算法的一个特例，即将整个群体都作为邻域。

6. 停止准则

一般使用最大迭代次数或可以接受的满意解作为停止准则。

7. 粒子空间的初始化

较好地选择粒子的初始化空间，将大大缩短收敛时间。这是问题依赖的。

从上面的介绍可以看到，实际上粒子群优化算法并没有过多需要调节的参数。相对来说，惯性权重和邻域定义较为重要。

7.2.4 计算举例

下面以一个简单的例子来说明粒子群优化算法是如何工作的。

1. 最优化问题

求解以下的无约束优化问题（Rosenbrock 函数）：

$$\min f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$
$$x \in [-30, 30]^n \tag{7.4}$$

其中，问题的维数 $n = 5$。

当变量取二维时目标函数的图形如图 7.1 所示。

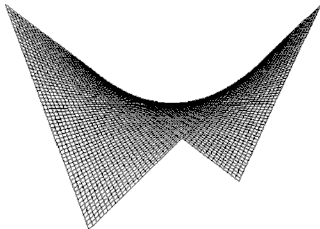


图 7.1 Rosenbrock 曲面图

2. 简单分析

Rosenbrock 是一个著名的测试函数，也叫香蕉函数，其特点是该函数虽然是单峰函数，在 $[100, 100]^n$ 上只有一个全局极小点，但它在全局极小点临近的狭长区域内取值变化极为缓慢，常用于评价算法的搜索性能。这种优化问题非常适合于使用粒子群优化算法来求解。这里使用标准版本的算法来求解，算法的相关设计分析如下。

编码：因为问题的维数为 5，所以每个粒子为 5 维的实数向量。

初始化范围：根据问题要求，设定为 $[-30, 30]$。根据前面的参数分析，可以将最大速度设定为 $V_{max} = 60$。

种群大小：为了说明方便，这里采用一个较小的种群规模，$m = 5$。

停止准则：设定为最大迭代次数 100 次。

惯性权重：采用固定权重0.5。

邻域拓扑结构：使用星形拓扑结构，即全局版本的粒子群优化算法。

3. 步骤

第1步：设置相关参数，在初始化范围内，对粒子群进行随机初始化，包括随机位置和速度。

第2步：计算每个粒子的适应值。

第3步：更新粒子的个体历史最好值和最好解以及整个群体的历史最好值和最好解。

第4步：根据式（7.3）和式（7.2）对粒子的速度和位置进行更新。

第5步：若迭代次数未达到100，则转第2步。

4. 一次迭代结果

各个粒子的初始位置如下：

$\boldsymbol{x}_1^0=(-15.061812,-23.799465,25.508911,4.867607,-4.6115036)$

$\boldsymbol{x}_2^0=(29.855438,-25.405956,6.2448387,10.079613,-26.621386)$

$\boldsymbol{x}_3^0=(23.805588,19.57822,-8.61554,9.441231,-29.898735)$

$\boldsymbol{x}_4^0=(7.1804657,-13.258207,-29.63405,-27.048172,2.2427979)$

$\boldsymbol{x}_5^0=(-4.7385902,-17.732449,-24.78365,-3.8092823,4.3552284)$

各个粒子的初始速度如下：

$\boldsymbol{v}_1^0=(-5.2273927,15.964569,-11.821243,42.65571,-48.36218)$

$\boldsymbol{v}_2^0=(-0.42986897,-0.5701652,-18.416643,-51.86605,-33.90133)$

$\boldsymbol{v}_3^0=(13.069403,-48.511078,28.80003,-8.051167,-28.049505)$

$\boldsymbol{v}_4^0=(-8.85361,12.998845,-13.325946,18.722532,-26.033237)$

$\boldsymbol{v}_5^0=(-5.7461033,-7.451118,29.135513,-14.144024,-41.325256)$

各个粒子的初始适应值如下：

$f_1^0=7.733296\text{E}7$

$f_2^0=1.26632864\text{E}8$

$f_3^0=4.7132888\text{E}7$

$f_4^0=1.39781552\text{E}8$

$f_5^0=4.98773\text{E}7$

显然，粒子3取得了群体中最好的位置和适应值，将其作为群体历史最优解。

经过一次迭代后，粒子的位置变化为

$\boldsymbol{x}_1^1=(2.4265985,29.665405,18.387815,29.660393,-39.97371)$

$x_2^1 = (22.56745, -3.999012, -19.23571, -16.373426, -45.417023)$

$x_3^1 = (30.34029, -4.6773186, 5.7844753, 5.4156475, -43.92349)$

$x_4^1 = (2.7943296, 19.942759, -24.861498, 16.060974, -57.757202)$

$x_5^1 = (27.509708, 28.379063, 13.016331, 11.539068, -53.676777)$

从上面的数据可以看到，粒子有的分量跑出了初始化范围。需要说明的是，在这种情况下，一般不强行将粒子重新拉回到初始化空间，即使初始化空间也是粒子的约束空间。因为，即使粒子跑出初始化空间，随着迭代的进行，如果在初始化空间内有更好的解存在，那么粒子也可以自行返回到初始化空间。

而且有研究表明，即使将初始化空间不设定为问题的约束空间，即问题的最优解不在初始化空间内，粒子也可能找到最优解。

第一次迭代后，各个粒子的适应值为

$f_1^1 = 1.68403632E8$

$f_2^1 = 5.122986E7$

$f_3^1 = 8.6243528E7$

$f_4^1 = 6.4084752E7$

$f_5^1 = 1.21824928E8$

此时，取得最好解的是粒子 2。

5. 100 次迭代结果

100 次迭代后，粒子的位置及适应值如下：

$x_1^{100} = (0.8324391, 0.71345127, 0.4540729, 0.19283025, -0.01689619)$

$x_2^{100} = (0.7039059, 0.75927746, 0.42355448, 0.20572342, 1.0952349)$

$x_3^{100} = (0.8442569, 0.6770473, 0.45867932, 0.19491772, 0.016728058)$

$x_4^{100} = (0.8238968, 0.67699957, 0.45485318, 0.1967013, 0.015787406)$

$x_5^{100} = (0.8273693, 0.6775995, 0.45461038, 0.19740629, 0.01580313)$

$f_1^{100} = 1.7138834$

$f_2^{100} = 121.33863$

$f_3^{100} = 1.2665054$

$f_4^{100} = 1.1421927$

$f_5^{100} = 1.1444693$

从结果可以看到，粒子 2 的适应值较大，这是因为 100 次迭代后粒子群还没有充分收敛。而这也在一定程度上保持了种群的多样性。图 7.2 是群体历史最优适应值随迭代次数增加的变化曲线。因为适应值变化过大，所以对其取对数。

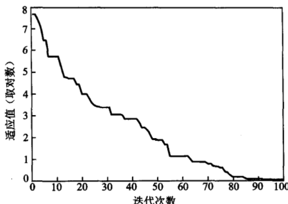


图7.2 适应值曲线

7.3 PSO的改进与变形

这一节里，首先介绍算法的三个构成要素的选择和调节：惯性权重、邻域拓扑结构和学习因子；然后介绍粒子群优化算法另外一个重要的改进版本：带有收缩因子的粒子群优化算法；再针对离散优化问题，说明两个典型的离散版本粒子群优化算法；之后，介绍几种基于遗传思想和梯度信息的改进策略；最后是算法在两类复杂环境中的解决方案：约束优化和多目标优化。

7.3.1 惯性权重

惯性权重是粒子群优化算法标准版本的重要参数，算法的成败很大程度上取决于该参数的选取和调节。下面介绍设置惯性权重的几种基本方法。

1. 固定权重

即赋予惯性权重以一个常数值，一般来说，该值在0和1之间。固定的惯性权重使粒子在飞行中始终具有相同的探索和开发能力。显然，对于不同的问题，获得最好优化效果的这个常数是不同的，要找到这个值需要大量的实验。通过实验发现：种群规模越小，需要的惯性权重越大，因为此时种群需要更好的探索能力来弥补粒子数量的不足，否则粒子极易收敛；种群规模越大，需要的惯性权重越小，因为每个粒子可以更专注于搜索自己附近的区域。

2. 时变权重

一般来说，希望粒子群在飞行开始的时候具有较好的探索能力，而随着迭代次数的增加，特别是在飞行的后期，希望具有较好的开发能力。所以希望动态调节惯性权重。可以通过时变权重的设置来实现。设惯性权重的取值范围为：[ω_{min}，ω_{max}]，最大迭代次数为 $Iter_max$，则第 i 次迭代时的惯性权重可以取为

$$\omega_i = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{Iter_max} \times i \tag{7.5}$$

这是一种线性减小的变化方式。也可以采用非线性减小的方式来设置惯性权重。根据实际的问题来确定最大权重 ω_{max} 和最小权重 ω_{min}。线性时变权重是在实际应用中使用最为广泛的一种方式。

3. 模糊权重

模糊权重是使用模糊系统来动态调节惯性权重。例如，文献 [28] 使用如下具有 9 条规则、2 个输入和 1 个输出的模糊系统。

输入变量：两个。当前最好的适应值（*CBPE*）和当前惯性权重。

输出变量：一个。即惯性权重的变化（百分比表示）。

这里，*CBPE* 测量的是 PSO 找到的最好候选解的性能。由于不同的优化问题有不同的性能评价值范围，所以为了让该模糊系统有广泛的适用性，可以使用标准化的 *CBPE*（*NCBPE*）。假定优化问题为最小化问题，则

$$NCBPE = \frac{CBPE - CBPE_{min}}{CBPE_{max} - CBPE_{min}} \tag{7.6}$$

其中，$CBPE_{min}$ 为估计的（或实际的）最小值，而 $CBPE_{max}$ 为非优 *CBPE*，任何 *CBPE* 值大于或等于 $CBPE_{max}$ 的解都是最小化问题所不能接受的解。

每个输入和输出定义了三条模糊集合：低、中、高，相对应的隶属度函数分别为：左三角形、三角形和右三角形。共九条规则。这三个隶属度函数分别定义如下。

左三角隶属度函数

$$f_{left_triangle} = \begin{cases} 1 & x < x_1 \\ \dfrac{x_2 - x}{x_2 - x_1} & x_1 \leqslant x \leqslant x_2 \\ 0 & x > x_2 \end{cases} \tag{7.7}$$

三角隶属度函数

$$f_{\text{triangle}}=\begin{cases}0 & x<x_1\\ 2\dfrac{x-x_1}{x_2-x_1} & x_1\leqslant x\leqslant\dfrac{x_2+x_1}{2}\\ 2\dfrac{x_2-x}{x_2-x_1} & \dfrac{x_2+x_1}{2}<x\leqslant x_2\\ 0 & x>x_2\end{cases}\tag{7.8}$$

右三角隶属度函数

$$f_{\text{right_triangle}}=\begin{cases}0 & x<x_1\\ \dfrac{x-x_1}{x_2-x_1} & x_1\leqslant x\leqslant x_2\\ 1 & x>x_2\end{cases}\tag{7.9}$$

这里，x_1 和 x_2 是决定隶属度函数形状的关键参数。

4. 随机权重

随机权重是在一定范围内随机取值。例如可以取值如下

$$\omega=0.5+\frac{Random}{2}\tag{7.10}$$

其中，*Random* 为 0～1 之间的随机数。这样，惯性权重将在 0.5～1 之间随机变化，均值为 0.75。之所以这样设定，是为了应用于动态优化问题。将惯性权重设定为线性减小的时变权重是为了在静态的优化问题中使粒子群在迭代开始的时候具有较好的全局寻优能力，即探索能力，而在迭代后期具有较好的局部寻优能力，即开发能力。而对于动态优化问题来说，不能够预测在给定的时间粒子群需要更好的探索能力还是更好的开发能力。所以，可以使惯性权重在一定范围内随机变化。

7.3.2　邻域拓扑结构

如何定义粒子的邻域组成，即邻域的拓扑结构，是算法实现中的一个基本问题。在 7.2.1 中介绍了，一般有两种方式组成邻域，一种是索引号相邻的粒子组成邻域，另一种是位置相邻的粒子组成邻域。下面来详细说明这两类邻域拓扑结构。

1. 基于索引号的拓扑结构

这类拓扑结构最大的优点是在确定邻域时不考虑粒子间的相对位置，从而避免确定邻域时的计算消耗。

（1）环形结构

环形结构是一种基本的邻域拓扑结构，每个粒子只与其直接的 *K* 个邻居相连，即，与该粒子索引号相近的 *K* 个粒子构成该粒子的邻域

成员。例如，当 $K=2$ 时，对于粒子 i，定义其邻域成员为：粒子 $i-1$ 和粒子 $i+1$（也可以将上述情况称为邻域半径为 1）。在迭代过程中，这种邻域组成保持不变。图 7.3 是环形拓扑结构的示意图。

环形结构下，种群的一部分可以聚集于一个局优，而另外一部分可能聚集于不同的局优，或者再继续搜索，避免过早陷入局优。邻居间的影响一个一个地传递，直到最优点被种群的任何一个部分找到，然后使整个种群收敛。

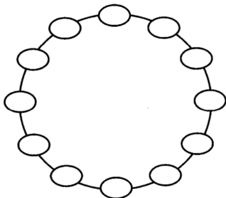


图 7.3 环形拓扑结构

可以在环形拓扑结构中加入两条捷径（Shortcut），得到带有捷径的环形拓扑结构，如图 7.4 所示。有两个粒子的邻域发生变化，即随机地选取种群中的另一个粒子作为自己的邻域成员，从而加强了不同粒子邻域之间的信息交流。这样变化后的环形拓扑结构缩短了邻域间的距离，种群将更快收敛。

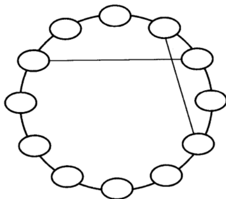


图 7.4 带有捷径的环形拓扑结构

(2) 轮形结构

轮形结构是令一个粒子作为焦点，其他粒子都与该焦点粒子相连，而其他粒子之间并不相连，如图7.5所示。这样所有的粒子都只能与焦点粒子进行信息交流，有效地实现了粒子之间的分离。焦点粒子比较其邻域（即整个种群）中所有粒子的表现，然后调节其本身飞行轨迹向最好点靠近。这种改进再通过焦点粒子扩散到其他粒子。所以焦点粒子的功能类似一个缓冲器，减慢了较好的解在种群中的扩散速度。

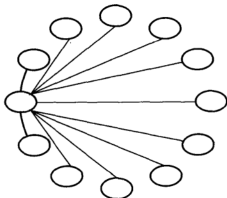


图7.5　轮形拓扑结构

同样也可以在轮形拓扑结构中加入两条捷径，得到带有捷径的环形拓扑结构，如图7.6所示。带有捷径的轮形拓扑结构可以产生两方面的效果。一方面，能够产生迷你邻域（Mini－Neighborhood），迷你邻域中的外围粒子被直接与焦点粒子相连的粒子所影响，这样在迷你邻域这个合作的子种群内可以更快地收敛，焦点粒子的缓冲器又可以防止整个种

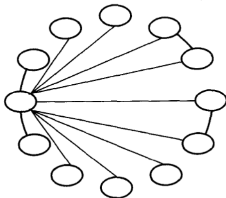


图7.6　带有捷径的轮形拓扑结构

群过早收敛于局优。另一方面，也可能产生孤岛，或分离子种群间的联系，使子种群内部进行合作，独立地进行问题的优化。这将导致信息交流的减少，使那些分离的个体不能得到整个种群所找的好的区域；也使种群其他粒子不能分享被分离的个体搜索中获得的成功信息。

（3）星形结构

星形拓扑结构是每个粒子都与种群中的其他所有粒子相连，即将整个种群作为自己的邻域。也就是粒子群算法的全局版本。这种结构下，所有粒子共享的信息是种群中表现最好的粒子的信息。

（4）随机结构

随机结构是在 N 个粒子的种群中间，随机地建立 N 个对称的两两连接。

2. 基于距离的拓扑结构

基于距离的拓扑结构是在每次迭代时，计算一个粒子与种群中其他粒子之间的距离，然后根据这些距离来确定该粒子的邻域构成。下面是一个具体的实现方法：动态邻域拓扑结构。

在搜索开始的时候，粒子的邻域只有其自己，即将个体最优解作为邻域最优解，然后随着迭代次数的增加，逐渐增大邻域，直至最后将群体中所有粒子作为自己的邻域成员。这样使初始迭代时可以有较好的探索性能，而在迭代后期可以有较好的开发性能。

对将要计算邻域的粒子 i，计算其与种群中其他所有粒子的距离。该粒子与粒子 $l(l \neq i)$ 的距离记为 $dist[l]$。最大的距离记为 max_dist。

定义一个关于当前迭代次数的函数 $frac$（取值为纯小数）

$$frac = \frac{3.0 \times ITER + 0.6 \times MAXITER}{MAXITER} \tag{7.11}$$

当 $frac < 0.9$ 时，满足下列条件的粒子构成当前粒子 i 的邻域，即 $\frac{dist[l]}{max_dist} < frac$。

当 $frac \geq 0.9$ 时，将种群中所有粒子作为当前粒子 i 的邻域。

7.3.3 学习因子

学习因子一般固定为常数，并且取值为2。但是也有研究者尝试了一些其他的取值和其他的设置方式。

1. c_1 和 c_2 同步时变

Suganthan 在实验中，参照时变惯性权重的设置方法，将学习因子设置如下：设学习因子 c_1 和 c_2 的取值范围为 $[c_{min}, c_{max}]$，最大迭代次

数为 $Iter_max$，则第 i 次迭代时的学习因子取为

$$c_1 = c_2 = c_i = c_{max} - \frac{c_{max} - c_{min}}{Iter_max} \times i \tag{7.12}$$

这是一种两个学习因子同步线性减小的变化方式，所以这里称之为同步时变。特别地，Suganthan 在实验中将参数设置为：$c_{max} = 3$，$c_{min} = 0.25$。但是发现，这种设置下，解的质量反而下降。

2. c_1 和 c_2 异步时变

Ratnaweera 等提出了另一种时变的学习因子设置方式，使两个学习因子在优化过程中随时间进行不同的变化，所以这里称之为异步时变。这种设置的目的是在优化初期加强全局搜索，而在搜索后期促使粒子收敛于全局最优解。这种想法可以通过随着时间不断减小自我学习因子 c_1，和不断增大社会学习因子 c_2 来实现。

（1）在优化的初始阶段，粒子具有较大的自我学习能力和较小的社会学习能力，这样粒子可以倾向于在整个搜索空间飞行，而不是很快就飞向群体最优解。

（2）在优化的后期，粒子具有较大的社会学习能力和较小的自我学习能力，使粒子倾向于飞向全局最优解。

具体实现方式如下：

$$c_1 = (c_{1f} - c_{1i})\frac{iter}{Iter_max} + c_{1i} \tag{7.13}$$

$$c_2 = (c_{2f} - c_{2i})\frac{iter}{Iter_max} + c_{2i} \tag{7.14}$$

这里，c_{1i}，c_{1f}，c_{2i}，c_{2f}为常数，分别为 c_1 和 c_2 的初始值和最终值。$Iter_max$ 为最大迭代次数，$iter$ 为当前迭代数。Ratnaweera 等在研究中发现，对于大多数标准如下设置优化效果较好：

$$c_{1i} = 2.5,\ c_{1f} = 0.5,\ c_{2i} = 0.5,\ c_{2f} = 2.5$$

需要说明的是，异步时变的学习因子应与线性减小的时变权重配合使用，效果较好。

7.3.4 带有收缩因子的粒子群优化算法

基本粒子群优化算法有两种重要的改进版本：加入惯性权重和加入收缩因子（Constriction Factor）。惯性权重的版本已成为标准版本，所以放在算法的基本原理中介绍，收缩因子版本放在本节介绍。

Clerc 在原始粒子群优化算法中引入可收缩因子的概念，并指出该因子对于算法的收敛是必要的。在带有收缩因子的粒子群优化算法中，

速度的更新方程如式（7.15）所示。

$$v_{id}^{k+1}=K[v_{id}^{k}+c_1\xi\ (p_{id}^{k}-x_{id}^{k})\ +c_2\eta\ (p_{gd}^{k}-x_{id}^{k})] \tag{7.15}$$

这里，K是c_1和c_2的函数，具体表达为式（7.16）。

$$K=\frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|},\ \varphi=c_1+c_2>4 \tag{7.16}$$

Clerc将参数取值为$c_1=c_2=2.05$，$\varphi=4.1$。于是可得$K=0.729$。

显然，如果将标准版本的粒子群优化算法取参数

$$\omega=0.729$$

$$c_1+c_2=0.729\times2.05=1.49445$$

则标准版本的粒子群优化算法与带有收缩因子的粒子群优化算法等价。即带有收缩因子的版本可以看作为标准版本算法的一个特例。Eberhart和Shi通过实验建议：

（1）如果使用带有收缩因子的粒子群优化算法，将最大速度V_{max}限定为X_{max}。

（2）如果使用带有惯性权重的粒子群优化算法，根据式（7.16）来选择ω，c_1，c_2的值。

7.3.5 离散版本的粒子群优化算法

应该说，粒子群优化算法非常适合于求解连续优化问题，而求解离散优化问题并不是该算法的优势所在，因为离散变量在经过粒子群优化算法的速度和位置更新方程的计算后，很可能不再保持为离散变量，如何解决这一问题是离散版本算法的难点所在。目前有一些研究者在努力寻求该算法在离散优化中的解决方案。下面介绍其中典型的两个解决方案：二进制编码和顺序编码。

1. 二进制编码

离散版本的粒子群优化算法最早是由Kennedy提出，采用二进制编码。即粒子的位置向量的每一位取值为**0**或者**1**。下面首先将二进制编码算法的粒子更新公式列于式（7.17）~式（7.19），然后再详细说明。

$$v_{id}^{k+1}=v_{id}^{k}+c_1\xi(p_{id}^{k}-x_{id}^{k})+c_2\eta(p_{gd}^{k}-x_{id}^{k}) \tag{7.17}$$

$$x_{id}^{k+1}=\begin{cases}\mathbf{1},\ random<S\ (v_{id}^{k+1})\\ \mathbf{0},\ 其他\end{cases} \tag{7.18}$$

$$S(v_{id}^{k+1})=\frac{1}{1+\exp(-v_{id}^{k+1})} \tag{7.19}$$

式（7.18）中的*random*是一个［0，1］区间内的均匀分布的伪随

机数。

显然，式（7.17）看上去与基本粒子群优化算法的速度更新公式（7.1）完全相同。（由于二进制编码算法提出于1997年，当时还没有带有惯性权重的粒子群算法，故没有采用式（7.3））。但是速度更新公式中，速度的意义和位置的取值发生了变化。

关于位置的三个变量 x_{id}^k、p_{id}^k、p_{gd}^k 仍然分别表示粒子在 k 次迭代时的位置、个体历史最优位置和邻域最优位置。但是这些变量的取值已经全部为 **0** 或者 **1**。

v_{id}^{k+1} 在这里不再表示 $k+1$ 次迭代时粒子飞行的速度。其意义在于，根据 v_{id}^{k+1} 的值来计算确定 x_{id}^{k+1} 取值为 **1** 或者为 **0** 的概率。这个概率由函数 $S(v_{id}^{k+1})$ 来表达。即不论 k 次迭代时粒子的位值为 **0** 还是 **1**，在 $k+1$ 次迭代时 x_{id}^{k+1} 的取值都以概率 $S(v_{id}^{k+1})$ 取 **1**，以概率 $1-S(v_{id}^{k+1})$ 取 **0**，这正是式（7.18）表达的意义。

$S(v_{id}^{k+1})$ 是 v_{id}^{k+1} 的函数。二进制版本的初始想法是将原来公式中的速度 v_{id}^{k+1} 作为粒子取值为 **1** 的概率，但是因为 v_{id}^{k+1} 在计算中很可能不能保证将其值限制在［0，1］区间，所以要将 v_{id}^{k+1} 进行变换，将其映射到［0，1］区间，函数 $S(v_{id}^{k+1})$ 正是用来完成此映射功能。$S(v_{id}^{k+1})$ 是一个S型函数，将 v_{id}^{k+1} 进行简单的运算后可以满足概率取值的需求。

在基本粒子群优化算法中，粒子的速度被限制在一个范围内，即最大速度 V_{max}。在二进制版本中，仍然需要这种限制，即 $|v_{id}^{k+1}|<V_{max}$。因为通过计算可以知道，当 $v_{id}^{k+1}>10$ 时，$S(v_{id}^{k+1})$ 的值将很小，导致 x_{id}^{k+1} 几乎一定取值为 **0**，这失去了以概率取值的意义。所以，要用 V_{max} 来进行一下限制。Kennedy 认为将 V_{max} 取值为6较好，此时 $S(v_{id}^{k+1})$ 取值范围在0.9975~0.0025之间。值得注意的是，在应用于连续空间的基本粒子群优化算法中，V_{max} 越大则粒子位置的改变，可能越大，粒子将具有更好的探索能力；而在应用于离散空间的二进制版本算法中，则正好相反，V_{max} 越大，可能导致变化的概率越小，因为取值为负的 v_{id}^{k+1}，可能绝对值很大，但是经过式（7.19）计算后得到的概率很小。

2. 顺序编码

Hu X 等提出了一种改进的粒子群优化算法来解决排列问题，因为其编码规则与遗传算法的顺序编码相同，这里将其称为顺序编码的粒子群优化算法。下面是一个编码的例子。

$$\boldsymbol{X}=(1\quad 2\quad 3\quad 4\quad 5\quad 6\quad 7)$$

这里编码长度为7，也是粒子的位值的变化范围。

在这种改进的离散版本的算法中，将速度也定义为粒子变化的概率，而速度的更新公式也保持不变。如果速度较大，则粒子更可能变化为一个新的排列序列。这里速度显然也要被加以限定，映射到［0，1］区间。粒子的更新过程说明如下，首先是粒子速度的规范化。设粒子位值变化范围为 n，粒子的速度为 v_i^{k+1}，则将其规范化为

$$Swap(v_{id}^{k+1}) = \frac{|v_{id}^{k+1}|}{n} \tag{7.20}$$

显然，$Swap(v_{id}^{k+1})$ 的取值范围在 0，1 之间，它决定了粒子 i 的编码是否产生一个交换。如果以该概率产生一个交换，则交换后粒子 i 第 d 位变化为邻域最好解相应的位值。这个过程可以表示为如图 7.7 所示。

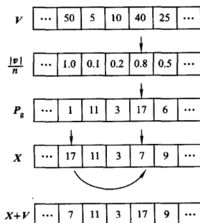


图 7.7 粒子更新示意图

图 7.7 中粒子位值变化范围为 50。速度分量为 40 的位置，粒子位值为 7，随机产生一个［0，1］区间内均匀分布的伪随机数，小于 0.8，所以该位置将产生一个交换，该位置邻域最优解 $\boldsymbol{P}_g$ 的值为 17，为了在交换后该位置取值与邻域最优解 $\boldsymbol{P}_g$ 保持一致，将该位置与粒子取值为 17 的位置交换。得到的结果为粒子的新的位置 $\boldsymbol{X}+\boldsymbol{V}$。

因为粒子以一定概率与邻域最优解的排列趋同，所以如果其与邻域最优解的排列相同时，将保持不变。为了避免这种情况，引入了变异来克服，即当粒子与邻域最优解排列相同时，随机选取编码中的两个位置，交换其位值，如图 7.8 所示。

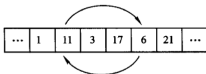


图 7.8 变异示意图

7.3.6 基于遗传策略和梯度信息的几种改进算法

对于大多数智能优化方法来说，遗传策略和梯度信息的使用是两类重要的改进方式。因为遗传算法是目前为止应用最为广泛的智能优化方法，其基本的遗传策略，包括选择、杂交和变异等已经深入人心并且取得了良好的优化效果，所以对于一种新的算法，研究者首先会想到用这样的策略来进行尝试，试图找到性能改进的措施。梯度信息是传统实优化中所使用的重要信息，或者说，传统实优化是基于梯度信息的。所以，对于粒子群优化算法来说，针对那些具有梯度信息的函数优化使用梯度信息，必然大大提高搜索效率。这里介绍一些基于遗传策略和梯度信息的改进粒子群优化算法。

1. 基于选择的改进算法

标准粒子群优化算法中，粒子的历史最优信息的确定相当于一种隐含的选择机制。在邻域拓扑结构中已经说明，这种选择机制可能通过较长时间才能发生作用。而传统的进化算法中选择的方法可以将搜索定向于较好的区域，合理地分配有限的资源。

Angeline 将自然选择机理与粒子群优化算法相结合，提出了一种混合群体算法（Hybrid Swarm）。该混合算法是采用的锦标赛选择方法（Tournament Selection Method），即每个粒子将其当前位置上的适应值与其他 k 个粒子的适应值比较，记下最差的一个得分，然后整个粒子群以分值高低排队。在此过程中，不考虑个体的历史最优值。群体排序完成后，用群体中最好的一半的当前位置和速度来替换最差的一半的位置和速度，同时保留原来个体所记忆的历史最优值。这样，每次迭代后，一半粒子将移动到搜索空间中相对较优的位置，这些个体仍保留原来的历史信息，以便于下一代的位置更新。

这种选择方法的加入使混合算法具有更强的搜索能力，特别是对于当前较好区域的开发能力，使得收敛速度加快。但是，增加了陷入局优的可能性。

2. 基于交叉的改进算法

LØvbjerg 等人基于进化计算中交叉的思想提出了带有繁殖（Breeding）和子种群（Subpopulations）的混合粒子群优化算法，这里研究者使用的是繁殖一词，与交叉同义。这种混合算法的结构如图 7.9 所示。

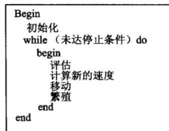


图 7.9 基于交叉的混合粒子群算法结构

这里，速度的计算方式采用的是惯性权重和收缩因子相结合的公式，如式（7.21）所示

$$v_{id}^{k+1} = k(\omega v_{id}^{k} + c_1\xi(p_{id}^{k} - x_{id}^{k}) + c_2\eta(p_{gd}^{k} - x_{id}^{k})) \tag{7.21}$$

移动（即新位置的计算）方式仍为式（7.2）。

繁殖的方式如下：每次迭代时，依据一定的概率（称为繁殖概率）在粒子群中选取一定数量的粒子放入一个池中，池中粒子随机两两进行繁殖，产生相应数目的子代粒子，并用子代粒子代替父代粒子，使种群规模保持不变。

每一维中子代位置由父代位置进行算术交叉计算得到

$$child_1(x_i) = p_i \times parent_1(x_i) + (1.0 - p_i) \times parent_2(x_i) \tag{7.22}$$

$$child_2(x_i) = p_i \times parent_2(x_i) + (1.0 - p_i) \times parent_1(x_i) \tag{7.23}$$

这里，p_i 是 0，1 之间均匀分布的伪随机数。子代的速度向量由父母速度向量之和归一化后得到

$$child_1(\boldsymbol{v}) = \frac{parent_1(\boldsymbol{v}) + parent_2(\boldsymbol{v})}{|parent_1(\boldsymbol{v}) + parent_2(\boldsymbol{v})|}|parent_1(\boldsymbol{v})| \tag{7.24}$$

$$child_2(\boldsymbol{v}) = \frac{parent_1(\boldsymbol{v}) + parent_2(\boldsymbol{v})}{|parent_1(\boldsymbol{v}) + parent_2(\boldsymbol{v})|}|parent_2(\boldsymbol{v})| \tag{7.25}$$

子种群的思想是将整个粒子群划分为一组子种群，每个子种群有自己的内部历史最优解。上述交叉操作可以在同一子种群内部进行，也可以在不同子种群之间进行。交叉操作和子种群操作，可以使粒子受益于父母双方，增强搜索能力，易于跳出局优。

3. 基于变异的改进算法

Higashi 和 Iba 将进化计算中的高斯变异融入粒子群的位置和速度的更新中，来避免陷入局优。每个粒子在搜索区域移动到另一位置时，不像标准公式那样只依据先验概率，不受其他粒子的影响，而是通过高斯变异加入了一定的不确定性。变异的计算公式为

$$mut(x) = x \times [1 + gaussian(\sigma)] \tag{7.26}$$

其中，$mut(x)$ 为变异后粒子的位置，取 σ 为搜索空间每一维长度的 0.1 倍，实验表明 σ 取此值时效果最好。算法以预定的概率来选择变异个体，并以高斯分布来确定它们的新位置。这样，在算法初期可以进行大范围的搜索，然后在算法的中后期通过逐渐减小变异概率来改进搜索效率，作者将变异概率设定为从 1.0 线性减小到 0.1。

基于高斯变异的混合算法更容易跳出局优，因此在求解多峰函数时表现出更好的性能。

Stacey 等人尝试了使用 Cauchy 分布来进行变异操作，其概率分布函数为

$$f(x) = \frac{a}{\pi} \cdot \frac{1}{x^2 + a^2} \tag{7.27}$$

其中，$a = 0.2$。Cauchy 分布与正态分布相似，但是在尾部有更大的概率，这样可以增加较大值产生的概率。

并且，将每个分量的变异概率设定为 $1/d$，这里 d 为向量的维数。

4. 带有梯度加速的改进算法

和其他进化算法一样，标准粒子群优化算法利用种群来进行随机搜索，没有考虑具体问题的特性，不使用梯度信息。而梯度信息往往包含目标函数的一些重要信息。

对于函数 $f(\boldsymbol{x}), \boldsymbol{x} = (x_1, x_2, \cdots, x_n)$，其梯度可以表示为

$\nabla f(\boldsymbol{x}) = [\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \cdots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}]^{\mathrm{T}}$，负梯度方向是函数值的最速下降方向。

王俊伟等通过引入梯度信息来影响粒子速度的更新，构造了一种带有梯度加速的粒子群优化算法，每次粒子进行速度和位置的更新时，每个粒子以概率 p 按照式（7.3）和式（7.2）进行更新；以概率 $(1-p)$ 按照梯度信息进行更新，在负梯度方向进行一次直线搜索来确定移动步长。梯度加速的流程具体如图 7.10 所示。

直线搜索采用了黄金分割法，这一步骤可以详述如下。

产生一个伪随机数，若大于 p，则按照式（7.3）和式（7.2）更新

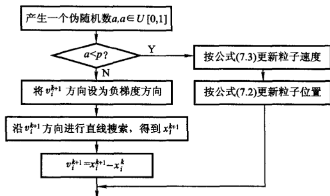


图 7.10 梯度加速的流程图

粒子速度和位置；否则，

①试探方式确定一单谷搜索区间 $\{\boldsymbol{a}, \boldsymbol{b}\}$。

②计算 $\boldsymbol{t}_2=\boldsymbol{a}+\beta(\boldsymbol{b}-\boldsymbol{a})$，$f_2=f(\boldsymbol{t}_2)$。

③计算 $\boldsymbol{t}_1=\boldsymbol{a}+\boldsymbol{b}-\boldsymbol{t}_2$，$f_1=f(\boldsymbol{t}_1)$。

④若 $|\boldsymbol{t}_1-\boldsymbol{t}_2|<\varepsilon$，（$\varepsilon$ 为终止限），则 $\dfrac{\boldsymbol{t}_1+\boldsymbol{t}_2}{2}$ 即为粒子的下一位置；否则转⑤。

⑤判别 $f_1 \leqslant f_2$ 是否满足，若满足，则置 $\boldsymbol{b}=\boldsymbol{t}_2$，$\boldsymbol{t}_2=\boldsymbol{t}_1$，$f_2=f_1$，然后转③；否则，若 $f_1>f_2$，则置 $\boldsymbol{a}=\boldsymbol{t}_1$，$\boldsymbol{t}_1=\boldsymbol{t}_2$，$f_1=f_2$，$\boldsymbol{t}_2=\boldsymbol{a}+\beta(\boldsymbol{b}-\boldsymbol{a})$，$f_2=f(\boldsymbol{t}_2)$，然后转④。

同时为了减小粒子陷入局优的可能性，对群体最优值进行观察。在寻优过程中，当最优信息出现停滞时，对部分粒子进行重新初始化，从而保持群体的活性。

梯度信息的加入使粒子的移动更有针对性，移动更有效率，进一步提高 PSO 算法的收敛速度，但是也会增加算法的问题依赖性，特别是有些问题的梯度信息极易将粒子引入局优。所以带有梯度加速的粒子群优化算法需要根据问题的性质来调整梯度信息对于粒子移动的影响程度。

7.3.7 约束的处理

智能优化方法处理约束的一般性策略都可以借鉴到粒子群算法中来，也可以根据粒子群优化的特性来设计专门的约束处理方式。下面是在粒子群优化算法中对约束处理方法的一个概述。详细内容可以参看本

章参考文献中的相关部分［10，11，22，25］。

1. 惩罚策略

粒子群优化算法解决约束问题，同样可以在目标函数中加入惩罚函数。在前面章节中已经说明过，这种方法的主要问题在于设计好惩罚函数。如 Parsopoulos 使用了非固定多阶段指派惩罚函数来解决约束问题。

2. 拒绝策略

在3.4.6节中已经说明，拒绝策略是抛弃进化过程中产生的所有不可行解。Hu 和 Eberhart 根据粒子群优化算法的特点又进行了修改，来保持解的可行性：粒子在整个空间进行搜索，但是只保持跟踪那些可行的解来加速它们的搜索过程（拒绝将非可行解作为历史信息），所有的粒子被初始化为可行解。具体的改进步骤可以描述为下面两条。

（1）初始化中，所有的粒子被重复地进行初始化，直到满足所有的约束。

（2）在计算并保留个体和邻域内的历史最好解时，只保留那些满足约束的解。

同在遗传算法中所讲的一样，上述方法的缺点在于可行的初始种群可能难以找到。

El－Galled 等也使用了类似的方法，但是在粒子飞出可行空间时，将粒子重新设置为过去找到的最好可行解的位置。该方法的缺点在于，可能导致粒子被限制在初始点区域。

3. 基于 Pareto 的方法

基于 Pareto 的方法源于解决多目标优化问题，近年来也有人使用该方法来解决约束优化问题。如 Ray 等在粒子群中使用了多阶段信息共享策略来处理单目标约束问题，就是利用 Pareto 排序来产生共享信息。

（1）将约束处理为约束矩阵，基于此矩阵，使用 Pareto 排序来产生表现好的粒子，放入好解列表（Better Performer List，BPL）中。

（2）BPL 之外的粒子在飞行时使用了离其最近的 BPL 中的粒子信息。即一个 BPL 中的粒子和其附近的非 BPL 粒子组成一个邻域。

（3）在飞行中参考邻域最好解（Leader）的信息时，使用了一个简单的进化算子代替常规公式来避免早熟。即变量值产生于粒子本身和 Leader 之间的概率为50%；变量值产生于变量值下限和粒子与 Leader 最小值之间的概率为25%；变量值产生于变量值上限和粒子与 Leader 最大值之间的概率为25%。

在本章7.4.1节的应用实例中，将详细说明一个处理约束问题的方法。

7.3.8 多目标的处理

应用粒子群优化算法解决多目标问题是近年来的研究热点之一。有的研究者借鉴传统的进化算法解决多目标的方案，如基于目标加权和向量评价方法。更多的研究是根据粒子群算法的特点，通过选取邻域最优解来开发基于记忆的方法。根据目前的研究情况，将这些方法分为两大类，传统方法和基于记忆的方法。下面进行概括性的介绍。

1. 传统方法

Parsopoulos 采用了两种传统的多目标处理方法来应用于粒子群优化算法求解多目标问题：权重和方法和向量评价法。具体方法和 3.4.7 节中所述是相同的。

权重和方法中，采用了传统线性加权方法、“Bang - Bang”加权方法和动态加权方法。

向量评价方法中，首先利用单目标优化函数的个体评价方法对粒子进行评估，从而形成不同的子种群，每个粒子均是某一目标函数的较优解。粒子在飞行中又受到其他子种群信息的影响，从而向满足其他目标函数分量的方向飞行，逐渐满足更多的目标函数分量，最终朝着 Pareto 最优方向飞行。

2. 基于记忆的方法

粒子群优化中信息的共享机制与其他基于种群的优化工具有很大不同。遗传算法中通过交叉实现信息的交换，这是一种双向的信息共享机制。而粒子群中只有邻域最好解将其信息给予其他粒子，这是一种单向的信息共享机制。由于点吸引的特性，传统粒子群不能同时向多个Pareto最优解靠近。虽然可以对多个目标使用不同的权重多次运行算法，找到 Pareto 最优解，但是最好的办法还是同时找到 Pareto 最优解集。

作为非独立的智能体，粒子飞行时个体和邻域内的历史最好解具有关键性的作用，所以解决多目标优化问题，个体和邻域最优解的选取是关键。对于个体历史信息，可以让粒子记忆更多的 Pareto 最优解的信息。目前的研究主要集中在邻域最优解的选取。邻域最优解的选取可以分为两个步骤。

（1）确定邻域

也就是确定邻域的拓扑结构，在 7.3.2 节中已经介绍了一些基本的邻域策略，在多目标处理中，也可以使用一些特殊的邻域策略。

（2）从邻域中选取一个表现好的粒子作为最优解

邻域最优解的选择应该满足两个原则：首先，能够为粒子群收敛提

供有效的导向作用；其次，能够在搜索 Pareto 解集和保持种群多样性之间保持平衡。文献中的方法主要包括两种。

①旋转法

根据某个规则赋予每个邻域中候选粒子以一个权重，然后用旋转法随机选择。这样能够保持种群多样性。

②定量法

使用一些具体的方法来定量计算得到邻域最优解，而不是随机选择。

关于多目标处理的详细内容可以参看本章参考文献中的相关部分[4，5，12，19，21，23，26]。

7.4 应用实例

本节选取了两个应用实例，即网络广告资源优化模型和新产品组合投入模型，来具体说明粒子群优化算法求解过程。网络广告资源优化问题的求解，主要是说明粒子群优化算法如何来解决约束问题；新产品组合投入模型的求解是用粒子群优化算法来解决离散优化问题。

7.4.1 网络广告资源优化

1. 问题的提出

这里，考虑门户网站旗帜类广告的资源优化问题。假设一个门户受理了 n 个广告业务，记为 $A_j(j=1,2,\cdots,n)$。网站的所有网页按属性分类，可以分为 m 类，用 $1,2,\cdots,i,\cdots,m$ 表示。例如某门户网站的网页可以划分为主页、新闻、体育、娱乐、游戏、旅游、文化等。定义单个网页的日访问量为：每天这个网页在不同 IP 地址用户的浏览器上出现的次数总和。可以用特定的软件统计属性 i 页面的平均日访问量，记为 k_i。d_{ij}表示广告 j 在网页 i 下的显示概率，c_{ij}表示在特定属性页面 i 下广告 j 的点击概率，c_{ij}可以通过统计历史数据得出。而 h_j 表示广告客户所要求的广告显示率。

2. 数学模型

下面介绍一个最大化广告效果函数的优化模型，优化配置广告资源。广告效果由广告点击率的大小指示。因为通常情况下，广告被点击意味着广告已呈现在用户面前，用户看到了他感兴趣的信息。为了避免线性规划中出现广告显示率过于集中在某几类页面，而增加广告与顾客的接触面，同时通过视觉刺激发挥广告树立品牌的作用。所以在最大化

广告总体点击次数的基础上加入了一个分散惩罚项，将广告显示率分散到不同的网页，

$$\max\left(\omega_1\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}k_id_{ij}-\frac{\omega_2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}k_i\sum_{i=1}^{m}\sum_{j=1}^{n}d_{ij}^2\right) \tag{7.28}$$

其中，目标函数中的第二项的最大值（即去掉负号后的最小值）是在各个 d_{ij} 相等时取得，所以第二项起到了将广告 i 分散于不同属性页的作用。两个权值 ω_1 和 $\omega_2(\omega_1+\omega_2=1)$，规定了第一项和第二项在目标函数中将起多大作用。同时满足下面的约束：

$$\text{s.t.}\quad \sum_{i=1}^{m}k_id_{ij}=h_j\quad j=1,\cdots,n$$

$$\sum_{j=1}^{n}d_{ij}=1\quad i=1,\cdots,m \tag{7.29}$$

$$d_{ij}\geqslant 0 \tag{7.30}$$

约束方程的含义为：①广告在各种属性页面的印次总数达到广告主的要求；②所有广告在各个属性页面的显示概率之和等于1；③显示概率非负。当约束方程满足 $\sum_{j=1}^{n}h_j=\sum_{i=1}^{m}k_i$ 时，问题存在可行解。此模型等价于非线性运输问题。

3. 使用粒子群优化算法求解

本文模型中约束为线性方程组，可行域狭小，因此难以设计一个合适的惩罚函数。如果惩罚较大，则惩罚后的目标函数将不连续；惩罚较小时，得到的最优解可能在可行集外。并且，算法的大部分时间耗费在从非可行域跳到可行域的运算中，运算效率低。因此，本文不用惩罚函数，而根据约束集和粒子群优化算法的特点，设计每一步迭代粒子都可行的粒子群优化算法。

(1) 粒子的初始化

首先，用矩阵表达决策变量 $d_{ij}(i=1,\cdots,m$ 且 $j=1,\cdots,n)$ 的集合，描述如下：

$$D=\begin{bmatrix}d_{11}&\cdots&d_{1n}\\ \vdots&&\vdots\\ d_{m1}&\cdots&d_{mn}\end{bmatrix}$$

粒子坐标就用 $D=\{d_{ij}\}$ 表示，描述粒子在 mn 维空间中的运动位置，而粒子的运动速度可由矩阵 $V=\{v_{ij}\}$ 表示。根据表上作业法中初始基可行解的确定方法，得到下面的随机初始化过程，产生满足约束条件即式（7.29）和式（7.30）的初始粒子。

```
初始化1：
  Input  Array  h[j], for j=1, …, n
         Array  a[i]=1 and k[i], for i=1, …, m
  Begin
设定集合 Ag={1,2,…,mn}
repeat
  从集合中随机选一个数 r;
  计算相应的行和列;
  i=Int((r-1)/(n+1));
  j=(r-1)mod(n+1);
  令 d[i][j]=min(a[i],h[j]/k[i]);
  修改数据
  a[i]=a[i]-d[i][j];
  h[j]=h[j]-d[i][j]k[i];
  集合 Ag=Ag\ {r};
until 集合 Ag 为空
End
```

初始化过程1产生一个满足约束条件的粒子 $D=\{d_{ij}\}$，这是一个至多有 $m+n-1$ 个非零元素的矩阵，表示凸可行解空间的顶点，对应约束集中的基可行解。

注：这里的初始化过程与表上作业法确定初始基可行解的方法区别是，初始化1是随机选择变量，而表上作业法根据最小运价或最大的运价差额（最小运价与次最小运价的差额）来选择变量。所以初始化1得到的是运输问题的基可行解，它对应可行解空间的顶点，可行域中的任意一点都能用顶点的凸组合来表示。

为了求解最优化问题如式（7.28）所示，需要设计可保持粒子可行性条件的速度。由问题可行集和粒子群优化算法中粒子速度、位置更新公式的特点，推导出以下定理。

定理7.1 如果可行粒子的初始速度 V 满足条件

$$\sum_{i=1}^{m} k_i v_{ij} = 0 \quad j=1,\cdots,n$$
$$\sum_{j=1}^{n} v_{ij} = 0, \quad i=1,\cdots,m \tag{7.31}$$

则按更新公式（7.2）迭代，得到的粒子满足约束条件即式（7.29）。

证明：设可行粒子 $D=\{d_{ij}\}(i=1,\cdots,m;j=1,\cdots,n)$，按式(7.2) 更新后得到粒子 $(d_{ij}+v_{ij})$，代入式 (7.29)，并且由式(7.31)，可得

$$\sum_{i=1}^{m} k_i(d_{ij}+v_{ij}) = \sum_{i=1}^{m} k_i d_{ij} = h_j \quad j=1,\cdots,n$$

$$\sum_{j=1}^{n} d_{ij}+v_{ij} = \sum_{j=1}^{n} d_{ij} = 1 \quad i=1,\cdots,m$$

所以更新后的粒子 $(d_{ij}+v_{ij})$ 满足约束条件即式 (7.29)，定理 7.1 得证。

定理 7.2 定义为 $v_{ij}=w(d_{1ij}-d_{2ij})(i=1,\cdots,m;j=1,\cdots,n,w\in R)$ 的速度使得式 (7.31) 成立，其中 d_{1ij}和 d_{2ij}分别是两个可行粒子 D_1 和 D_2 的位置分量。

证明：$v_{ij}=w(d_{1ij}-d_{2ij})$ 代入约束条件 (7.31) 左边，可得

$$\sum_{i=1}^{m} k_i v_{ij} = w\sum_{i=1}^{m} k_i d_{1ij} - w\sum_{i=1}^{m} k_i d_{2ij} = wh_j - wh_j = 0,\ j=1,\ \cdots,\ n$$

$$\sum_{j=1}^{n} v_{ij} = w\sum_{j=1}^{n} d_{1ij} - w\sum_{j=1}^{n} d_{2ij} = w - w = 0, \qquad i=1,\cdots,m$$

因此，$V=\{v_{ij}\}$满足式 (7.31)，定理 7.2 得证。

由定理 7.1 和定理 7.2 可得下面的推论。

推论 7.1 如果每个粒子的各个分量都采用相同的 ξ 和 η，并且初始速度满足式 (7.31)，则按照式 (7.3) 和式 (7.2) 更新的粒子总是满足约束条件即式 (7.29)。

因此，为了在每一步迭代中都获得满足约束条件即式 (7.29) 的粒子群，不但要产生满足约束的可行粒子，还需产生满足式 (7.31) 的初始速度。由定理 7.2，可以通过初始化过程 1 产生可行粒子，来获得满足式 (7.31) 的初始速度。

初始化 2：

Begin

运用初始化 1，得到一群可行粒子（至少为 mn 个）

Repeat

随机从粒子种群中取出两个粒子 $D[p]$ 和 $D[q]$，

令 $V[k]=(D[p]-D[q])/(g*rand()+0.2)$，其中 p，$q\in\{1,\cdots,Num\}$，$g\in R$，(根据具体问题确定)

$k=k+1$;

until $k>Num$（粒子群体规模）

End

其中，$rand$ ( ) 为 [0, 1] 区间内的伪随机数。

(2) 算法步骤

第1步：运用初始化1初始化一群粒子（群体规模为 Num）d_{ij}；并且运用初始化2，得到初始速度，设定初始个体历史最优解 p_{best} 和群体历史最优解 g_{best}（适值和位置同时设定）。注意在初始化2中得到了一群新的粒子。

第2步：计算每个粒子的适值。

第3步：对于每个粒子，将其适应值与其经历过的最好位置 p_{best} 作比较，并且检验这个粒子的可行性条件式（7.30），如果这个粒子的适值较好且满足 $d_{ij} \geq 0$，则将其作为当前的最好位置 p_{best}（粒子的适值和位置同时设定）。

第4步：对于每个粒子，将其适应值与全局所经历的最好位置 g_{best} 作比较，并且检验粒子的可行性条件式（7.30），如果这个粒子适值较好且满足 $d_{ij} \geq 0$，则将其设为 g_{best}（粒子的适值和位置同时设定）。

第5步：根据式（7.3）变化粒子的速度，为满足约束即式（7.29）令每个粒子各个分量的 ξ 和 η 值相等。然后设定一个小的正数 $\varepsilon = 10^{-6}$。如果一个粒子的速度小于 ε 并且在20步的迭代中仍保持小于 ε，则重新初始化速度和位置并重新设定 p_{best} 为当前的 d_{ij}，而保持 g_{best} 不变。然后根据式（7.2）变化粒子的位置。

第6步：如未达到结束条件（通常为足够好的适应值或达到一个预设最大代数 G_{max}），则返回第2步。

上述算法与标准粒子群优化算法区别如下。

(1) 对于约束的处理，首先产生可行的初始粒子和可行速度，保证每次迭代的粒子群满足约束条件即式（7.29）；其次，为了同时满足约束条件即式（7.30），在第3步和第4步比较适值时，加入了可行条件即式（7.30）的判断，只有当粒子同时满足适值较好和 $d_{ij} \geq 0$ 时，才将此粒子存入 p_{best} 或 g_{best}，这样就避免了粒子向适值较好的非可行解方向运动。

(2) 加入动态调整机制。通常当粒子群运行到一定的阶段，将会收敛到稳定状态，此时所有粒子将聚集到一起，粒子的调整速度越来越小，逐渐趋于零。所以此时应重新初始化速度和位置，令粒子恢复活力，发挥寻优的作用。

4. 计算实例

算法用JDK1.3实现，在P4　1.8 G，256 M计算机上对多个问题进行了成功的仿真，证明了算法的有效性。下面以规模为10个属性页，

20个广告的问题为例。随机产生20组数据，每组数据运算50次。其中，目标函数中的参数 γ 取值为0.02（若太大，就成为目标函数中的主要起作用部分了），算法参数设为：$\omega=0.5$，$c_1=0.7$，$c_2=0.7$。

第 k 组数据近优值中最好的记为 opt_{max}^k，最差的记为 opt_{min}^k，第 k 组数据第 l 次计算得到的目标值记为 opt_l^k。

图7.11中，1表示 $0\leqslant\varepsilon\leqslant0.05$ 的计算结果占全部计算结果的百分比；2表示 $0.05<\varepsilon\leqslant0.1$ 所占的百分比；3表示 $0.1\leqslant\varepsilon\leqslant0.25$ 所占的百分比；4表示 $0.25<\varepsilon$ 所占的百分比。

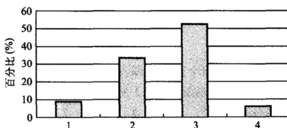


图7.11 误差结果分布图

（注：计算代数为10 000，粒子的种群规模为100）

计算 $\varepsilon=(opt_{max}^k-opt_l^k)/(opt_{max}^k-opt_{min}^k)$ 作为所得近优解的误差，计算所有近优解的误差，并统计误差在0～5%、5%～10%、10%～25%与大于25%各占的比率，标示如图7.11所示。误差基本在25%之内，说明计算的结果大部分集中在最好值附近。另外，计算20组数据 opt_{min}^k/opt_{max}^k 的比值的平均值为0.909，标准差为2.262 7，说明最差值与最好值的偏差为10%左右。所以总的来说算法的优化效果还比较好。

同时仿真结果也表明当计算的代数较大（>10 000），种群规模较大时，算法的效果较好。由于变量较多，基可行解空间的基数目（为171）较大，所以种群的规模较大时，算法才能发挥其效力。

限于篇幅，取一组数据下得到的最优结果和达优率列于表7.1和表7.2。

表7.1 优化结果

粒子个数	迭代次数		
	1 000	5 000	10 000
20	822.3	833	860.45
100	852.12	860.61	861.08

表 7.2 达优率（%）

粒子个数	迭代次数		
	1 000	5 000	10 000
20	32.40	49.00	91.20
100	81. 80	91. 40	93. 01

7.4.2 新产品组合投入问题

下面首先来介绍一个非线性半无限规划的相关新产品组合投入模型，然后用粒子群优化算法来求解。

1. 新产品投入模型

(1) 产品收益曲线

产品的生命周期分为四个阶段，即投入期、增长期、成熟期和衰退期。对于产品 i，其收益函数如图 7.12 所示，可表达为

$$f_i(t) = a_i t^2 e^{-t/b_i},\ t \in [0,\ T],\ i = 1,\ 2,\ \cdots,\ n \tag{7.32}$$

其中 a_i 和 b_i，$i=1$，2，…，n，为待定参数。

给定下列市场预测值：

p_i——产品能够达到的最大销售额即峰值。

r_i——达到峰值时间。

就可确定参数 a_i 和 b_i 如下：

$$a_i = \frac{p_i e^2}{4 b_i^{\ 2}}$$

$$b_i = \frac{p_i}{2}$$

该函数较好地表达了产品生命周期的四个阶段。选择不同参数就可获得不同类型的产品。

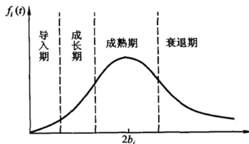


图 7.12 新产品的收益曲线

（2）新产品组合投入模型

变量和函数定义如下：

n——待投入新产品个数。

c_i——第 i 种产品（$i=1, 2, \cdots, n$）的投入成本。

t——时间变量。

$f_i(t)$ ——第 i 种产品的收益曲线。

$[d_{i1}, d_{i2}]$ ——第 i 种产品的最佳投入期。

α——产品在最佳投入期之前投入时的单位时间提前惩罚率（可以理解为投资贴现率）。

β——产品在最佳投入期之前投入时的单位时间的拖期惩罚率。

r_{ij}——产品之间的相关系数（相关收益率）（$|r_{ij}|<1$）。

r_{ij}的含义如下：

$r_{ij}>0$ 时，表示产品 i, j 是互补产品；

$r_{ij}=0$ 时，表示两个产品不相关；

$r_{ij}<0$ 时，表示产品 i, j 是互为替代产品，且有 $r_{ji}=r_{ij}$。

T——企业制定新产品投入的计划期，即 t 的变化区间为 $[1, T]$。

x_i——产品 i 的投入时间（为整形变量），其定义如下：

$$x_i=\begin{cases}N_i\in[1, T]，\text{产品 } i \text{ 在时刻 } N_i \text{ 投入}\\ 0，\text{产品 } i \text{ 不在计划期内投入}\end{cases} \tag{7.33}$$

并定义向量

$$\boldsymbol{X}=[x_1,x_2,\cdots,x_n]$$

及符号函数

$$\operatorname{sgn}(x)=\begin{cases}1, & x>0\\ 0, & x\leqslant 0\end{cases} \tag{7.34}$$

考虑到拖期惩罚，产品 i 的实际收益曲线为

$$ft_i(t)=[1-\beta(x_i-d_{i2})^+]f_i(t-x_i) \tag{7.35}$$

其中

$$(x-y)^+=\operatorname{sgn}(x-y)$$

定义下面的符号函数

$$Sr(x_i,x_j,t)=\operatorname{sgn}(x_i)\operatorname{sgn}(x_j)\operatorname{sgn}(t-x_i)\operatorname{sgn}(t-x_j) \tag{7.36}$$

则产品间相关收益为

$$fr(t,X)=\sum_{i<j\leqslant n}Sr(x_i,x_j,t)r_{ij}(ft_i(t)+ft_j(t)) \tag{7.37}$$

函数 $S(t)$ 和 $p(t)$ 的说明如下：

$S(t)$：企业的目标收益曲线。

$p(t)$：现有产品的收益曲线。

其中，$S(t)$ 的定义为

$$S(t)=s(1+r)^t$$

式中，s 为当年收益目标，r 为增长率。

因为开始投入时间定在现有产品的成熟期，现有产品的收益曲线可表示为

$$p(t)=a\exp(-t^2/2b^2)/(b\sqrt{2\pi})$$

如果用 p 表示现有产品当年利润额，q 表示 T 年内总利润额的预测值，很容易得出

$$a=2q, b=2q/(p\sqrt{2\pi})$$

令

$$g(t)=S(t)-p(t)$$

并考虑到提前惩罚，则组合投入模型为

$$\min \sum_{i=1}^{n} \operatorname{sgn}(x_i)[1+\alpha(d_{i1}-x_i)^+]c_i \tag{7.38}$$

$$\text{s.t.} \sum_{i=1}^{n} \operatorname{sgn}(x_i)ft_i(t)+fr(t,X)\geqslant g(t), \forall t\in[1,T] \tag{7.39}$$

$$x_i\in\{0,1,2,\cdots,T\}, i=1,2,\cdots,n \tag{7.40}$$

式(7.38)～式(7.40)是一个非线性的半无限规划模型。当 t 在区间$[1,T]$中取无限个值时，式(7.39)有无限个约束。下面使用粒子群优化算法来求解。

2. 粒子群优化算法结构

(1) 编码方法

本问题中编码方法采用投入时间向量作为粒子表达方式，即

$$\boldsymbol{X}=[x_1,x_2,\cdots,x_n]$$

代表一个粒子。其中，x_i 由式(7.33)定义，并且 x_i 允许重复。

(2) 评价函数

采用动态标定方法：

设 $F_{\max}$ 为本代粒子中目标函数的最大值，将目标函数转化为

$$z_{\max}=\max\{F_{\max}-\sum_{i=1}^{n} \operatorname{sgn}(x_i)[1+\alpha(d_{i1}-x_i)^+]c_i\}$$

为满足约束，取罚函数

$$h(X)=\min_{t\in[0,T]}\left[\sum_{i=1}^{n} \operatorname{sgn}(x_i)ft_i(t)+fr(t,X)-g(t)\right]$$

则评价函数为

$$F(X)=\{z_{max}+Mh(X),\ 0\} \tag{7.41}$$

其中，M 为一个随迭代次数变化的参数，迭代开始时 M 较小，即对不可行解的惩罚较小，可以保证粒子的多样性；随着迭代数的增加，M 逐渐增大，即对不可行解的惩罚加大，保证算法向最优解方向收敛。

（3）粒子运动公式

本算法中的粒子向量是由整数构成的，所以采用下列公式对粒子操作：

$$v_{id}=v_{id}+\begin{cases}Int(c_1r_1(p_{id}-x_{id})+c_2r_2(p_{gd}-x_{id})), & r_0\leqslant c_0\\ Int(Random\ (0,\ T)), & r_0>c_0\end{cases} \tag{7.42}$$

$$x_{id}=Mod(\mid x_{id}+v_{id}\mid) \tag{7.43}$$

其中，c_0 为［0，1］之间的常数，用来确定采取传统运动或是随机扰动。学习因子 c_1，c_2 是非负常数，r_0，r_1，r_2 为介于［0，1］之间的随机数。Int（·）为取整函数，$Random$（0，T）为取 0 ~ T 之间的随机数，是一个随机扰动，能够起到遗传算法中变异的作用。这里整数 T 表示计划期。Mod（·）表示以 T 为模求余数，其目的是防止粒子向量元素超出边界，函数中的绝对值是保证非负性。

式（7.42）的含义为：当产生的随机数小于 c_0 时进行如下操作：

$$v_{id}=v_{id}+Int(c_1r_1(p_{id}-x_{id})+c_2r_2(p_{gd}-x_{id})) \tag{7.44}$$

否则

$$v_{id}=v_{id}+Int(Rondom(0,T)) \tag{7.45}$$

（4）最优选择与终止准则

保存历史最好解，指定最大代数作停止准则。即选一个大的正整数 NG 为最大的代数，若迭代指标 k 大于 NG，则停止迭代并输出历史最好解作为最终的结果。

3. 算法步骤

算法步骤如下：

第 1 步：设置终止代数 NG、粒子群规模 m、学习因子 c_0，c_1，c_2；置 $k=0$；随机生成初始粒子群 $POP(0)=\{X^1,X^2,\cdots,X^m\}$；设置初始历史最好解 $X^{(*)}$，及初始历史最优目标值 $F(^*)$。

第 2 步：计算每个粒子的评价函数值，比较得到粒子自身经历过的最好位置 P_i，及全局经历过的最好位置 P_g。

第 3 步：按运动公式（7.42）进行上述提到的粒子操作。

第 4 步：如果 $k<NG$，$k=k+1$；否则，结束计算，输出全局历史最好解。

4. 仿真实例

设15个待投入产品，考虑在未来12个季度（3年）中投入。如果通过市场预测得到如表7.3所示的结果，为计算方便，所有数值均不计单位。

表7.3 待投入新产品的参数

序号	峰值（$4a_ib_i^2e^2$）	峰值时间（$2b_i$）	最早投入时间（d_{i1}）	最迟投入时间（d_{i2}）	投入成本（c_i）
1	180	2.5	1	7	90.0
2	130	5	2	6	170.0
3	155	2	2	9	127.5
4	166	1.4	1	5	75
5	115	2.3	6	10	20
6	100	1.3	3	8	40
7	160	2.8	3	7	50.5
8	175	1.5	9	12	40
9	95	4.3	3	8	90
10	60	2.2	5	9	40
11	120	1.8	1	6	50.0
12	165	2.5	2	6	88
13	155	1.4	4	9	76.5
14	175	1.1	5	8	65
15	185	2.3	3	8	100

现有产品投入当季度的利润额为180，T时间内总利润额的预测值为1 000；企业当季度的目标收益为180，季度增长率为0.1。

去掉全是0的行和列，简化和产品相关系数矩阵如表7.4所示，其中第一列和第一行表示产品号。

表7.4 产品相关系数阵

产品序号	1	4	5	9	15
1	0	0.25	−0.15	−0.1	0.21
4	0.25	0	−0.1	−0.10	0.23
5	−0.15	−0.1	0	0.18	−0.12
9	−0.1	−0.10	0.18	0	−0.12
15	0.21	0.23	−0.12	−0.12	0

设提前惩罚系数 $\alpha=0.1$，拖期惩罚系数 $\beta=0.01$。

给定 $NG=100$，$c_0=0.9$，$c_1=c_2=2$，及粒子群规模 $m=80$，经过几次运算，得到如表 7.5 所示的最好解。

表 7.5 计算结果

最好解	最优目标值
1 0 0 0 7 1 3 10 0 7 0 5 0 11 8	541.50

计算结果显示，在考虑了产品相关系数后，只有 2 个产品在最佳投入期之外投入，其中，产品 6 在最佳投入期之前投入，使总的投入成本增加，而产品 14 在最佳投入期之后投入，会损失一些市场份额或收益。但满足了企业的目标收益。其他产品都在最佳投入期之内投入。

实验结果表明，针对该模型设计的粒子群优化算法能够处理大规模非线性半无限规划，并且计算速度优于遗传算法；粒子群优化算法对求解组合优化问题是一个简单有效的方法。

问题与思考

1. 对于以下无约束优化问题：

$$\min f(\boldsymbol{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right]$$

其中，初值范围：$[-5.12, 5.12]^n$，$n=30$，目标达优值为 100。

试分别用遗传算法和粒子群优化算法设计其解决方案，并画出算法流程图。

2. 异步时变的学习因子一般与线性减小的时变惯性权重配合使用，在这种参数设置方式下，若学习因子 c_1 的变化范围为：［2.5，0.5］，c_2 的变化范围为：［0.5，2.5］。惯性权重的变化范围为：［0.9，0.4］。最大迭代次数为 4 000。试计算，当惯性权重为 0.8 和 0.5 时，两种学习因子分别如何取值？当时，粒子的探索和开发能力哪个占主导地位？粒子的两种学习能力哪个占主导地位？

3. 试为惯性权重设计一种非线性的变化方式，使其在粒子群搜索的初期以较慢的速度减小，在搜索的中期以较快的速度减小，而在搜索的后期又以较慢的速度减小。从而动态调节粒子群的探索和开发能力。

4. 粒子群优化算法和蚁群优化算法是群体智能的两个代表性算法，试分析其异同。

5. 试使用粒子群优化算法为旅行商问题设计一套解决方案。

参考文献

[1] Angeline P J. Using selection to improve particle swarm optimization:

IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1998: 84 - 89.

[2] Angeline P J. Evolutionary optimization versus particle swarm optimization: philosophy and performance defferences: Proc. of the 7th Annual Conf. on Evolutionary Programming [C]. Gemany: Springer, 1998: 601 - 610.

[3] Clerc M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization: IEEE Int. Congress. on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1999: 1951 - 1957.

[4] Coello Coello C A, Lechuga M S. MOPSO: a proposal for multiple objective particle swarm optimization: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2002: 1051 - 1056.

[5] Coello Coello C A, Pulido G T, Lechuga M S. Handling multiple objectives with particle swarm optimization [J]. IEEE transaction on evolutionary computation. 2004, 8 (3): 256 - 279.

[6] Eberhart R C, Kennedy J. A new optimizer using particle swarm theory. Proc. on 6th International Symposium on Micromachine and Human Science [C]. Piscataway, NJ: IEEE Service Center, 1995: 39 - 43.

[7] Eberhart R C, Shi Y. Particle swarm optimizatioon: developments, applications and resources: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2001: 81 - 86.

[8] Eberhart R C, Shi Y. Tracking and optimizing dynamic systems with particle swarms: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2001: 94 - 100.

[9] El-Gallad A I, El-Hawary M E, Sallam A A. swarming of intelligent particles for solving the nonlinear constrained optimization problem [J]. Engineering intelligent systems for electrical engineering and communications, 2001, 9 (3): 155 - 163.

[10] Higashi N, Iba H. Particle swarm optimization with gaussian mutation: Proc. of the swarm intelligence aymposium [C]. Piscataway, NJ: IEEE Service Center, 2003: 72 - 79.

[11] Hu X, Eberhart R. Solving constrained nonlinear optimization prob-

lems with particle swarm optimization: The 6th world multiconference on systemics, cybernetics and informatics [C]. 2002.

[12] Hu X, Eberhart R. Multiobjective optimization using dynamic neighborhood particle swarm optimization: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2002: 1677 - 1681.

[13] Hu X, Eberhart R and Shi Y. Swarm intelligence for permutation optimization: a case study of n-queens problem: IEEE swarm intelligences symposium [C]. Piscataway, NJ: IEEE Service Center, 2003: 243 - 246.

[14] Hu X, Shi Y, Eberhart R. Recent advances in particle swarm: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2004: 90 - 97.

[15] Kennedy J. Small world and mega-minds: effects of neighbourhood topology on particle swarm performance: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1999: 1931 - 1938.

[16] Kennedy J. The particle swarm: social adaptation of knowledge: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1997: 303 - 308.

[17] Kennedy J, Eberhart R C. Particle Swarm Optimization: Proc. IEEE International Conference on Neural Networks [C]. Piscataway, NJ: IEEE Service Center, 1995: 1942 - 1948.

[18] Kennedy J, Eberhart R C. A discrete binary of the particle swarm algorithm: Proc. IEEE International Conference on systems, man and cybernetics [C]. Piscataway, NJ: IEEE Service Center, 1997: 4104 - 4109.

[19] Li X. A non-dominated sorting particle swarm optimizer for multiobjective optimization: Lecture notes in computer science No. 2723: proceedings of the genetic and evolutionary computation conference [C]. 2003: 37 - 48.

[20] LØvbjerg M, Rasmussen T K, Krink T. Hybrid particle swarm optimizer with breeding and subpopulations: Proceedings of the Genetic and Evolutionary Computation Conference [C]. 2001: 469 - 476.

[21] Mostaghim S, Teich J. Strategies for finding local guides in multi-

objective particle swarm optimization: IEEE swarm intelligence symposium [C]. 2003: 26-33.

[22] Parsopoulos K E, Vrahatis M N. Particle swarm optimization method for constrained optimization problems: Euro-international sysposium on computational intelligence [C]. 2002: 1-7.

[23] Parsopoulos K E, Vrahatis M N. Particle swarm optimization method in multiobjective problems: ACM symposium on applied computing [C]. 2002: 26-33.

[24] Ratnaweera A, Halgamuge S K, Watson H C. Self-organizaing hierarchical particle swarm optimizer with time-varying acceleration coefficients [J]. IEEE transaction on evolutionary computation, 2004, 8 (3): 240-255.

[25] Ray T, Liew K M. A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimization problem: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2001: 75-80.

[26] Ray T, Liew K M. A swarm metaphor for multiobjective design engineering [J]. Engineering optimization, 2001, 34 (2): 141-153.

[27] Shi Y, Eberhart R. A modified particle swarm optimizer: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1998: 69-73.

[28] Shi Y, Eberhart R. Fuzzy adaptive particle swarm optimization: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2001: 101-106.

[29] Stacey A, Jancic M, Grundy I. Particle swarm optimization with mutation: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 2003: 1425-1430.

[30] Suganthan P N. Particle swarm optimization with neighbourhood operator: IEEE Int. Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1999: 1958-1962.

[31] 齐洁，汪定伟. 求解网络广告资源优化模型的改进微粒群算法 [J]. 控制与决策，2004，19 (8): 881-884.

[32] 万福才，汪定伟，李彦平. 微粒群优化算法在相关新产品组合投入的应用 [J]. 控制与决策，2004，19 (5): 520-524.

[33] 王俊伟，汪定伟. 带有梯度加速的粒子群算法 [J]. 控制与决

策，2004，19（11）：1298－1300.

［34］王俊伟，汪定伟．粒子群算法中惯性权重的实验与分析［J］．系统工程学报，2005，20（2）：194－198.

［35］谢晓锋，张文俊，杨之廉．微粒群算法综述［J］，控制与决策，2003，18（2）：129－133.

［36］曾建潮，介婧，崔志华．微粒群算法［M］．北京：科学出版社，2004.

第 8 章　捕食搜索算法

8.1 导　　言

捕食搜索（Predatory Search，简称 PS）算法是由巴西学者 Alexandre Linhares 在 1998 年提出来的一种用于解决组合优化问题的模拟动物捕食行为的空间搜索策略。Alexandre Linhares 把捕食搜索策略分别应用于解决旅行商问题（TSP）和超大规模集成电路设计（VLSI）问题，都取得了较好的效果。后来刘祟，汪定伟和蒋忠中等又将该算法进行了改进以及应用，都取得了较好的效果。

下面介绍捕食动物在搜索过程中的行为特征。

关于捕食动物的搜索行为的研究构成了例如生态学、生物学和动物行为学的主线。这些研究将捕食动物的搜索过程分为三个不同的部分：①首先，捕食动物必须为了捕食而搜索；②然后，捕食动物追逐和攻击猎物；③最后，处理并吃掉猎物。当然，不同的捕食动物在每一个步骤中需要不同的消耗。以狮子为例，因为它们捕食的是斑马或者瞪羚——形体较大容易被发现，在搜索的过程中不需要太多的消耗。对于这样的猎食者来说，它们在追逐和攻击阶段（攻击者可能有意外发生）以及进食阶段（其他动物可能参与分享猎物）需要更多的消耗。而对于鸟类或者蜥蜴等捕食小昆虫的动物来说，由于昆虫较小的形体（有时小于捕食者的 1%）很难被发现，搜索的过程需要很大的消耗。对于这些捕食动物来说，搜索阶段比攻击和处理阶段重要得多。自然选择的过程使这些动物进化出了有效的搜索策略。

对于搜索过程有重要需求的捕食动物来说，其最知名的搜索策略如下。在没有发现猎物和猎物的迹象时在整个捕食空间沿着一定的方向以很快的速度寻找猎物；一旦发现猎物或者发现有猎物的迹象，它们就立即改变自己的运动方式，减慢速度，不停地巡回，在发现猎物或者有猎物迹象的附近区域进行集中的区域搜索，持续不断地接近猎物。在搜寻一段时间没有找到猎物后，捕食动物将放弃这种集中的区域，而继续在整个捕食空间寻找猎物。这就是著名的区域限制（Area-Restricted）的搜索策略，在不同的物种如鸟类、蜥蜴、许多的捕食昆虫等行为中都有

描述。这一捕食行为看上去对于不同的环境和猎物分布都是适应的和有效率的。例如，如果猎物在搜索空间内是聚集的或者随机分布的，区域限制搜索能够通过在猎物附近进行持续的搜索从而最大化搜索成功的概率。如图 8.1 所示，动物的这种捕食过程中的搜索策略可以概括为以下两个搜索。

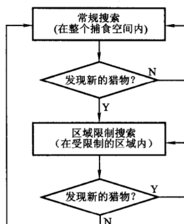


图 8.1 动物的捕食策略

搜索 1（常规搜索）：在整个搜索空间进行全面搜索，直到发现猎物或者有猎物的迹象而转到搜索 2 进行区域限制搜索。

搜索 2（区域限制搜索）：在猎物或者有猎物的迹象的附近区域进行集中搜索，直到搜索很多次也没有找到猎物而放弃局域搜索，转到搜索 1 进行常规搜索。

动物学家的进一步研究表明，动物的这种捕食搜索策略的效率是很高的。这是由于，除了简单和通用的特性，此策略很好地平衡了对整个猎物空间的探索（在整个解空间进行全面搜索）和对猎物聚集区域的开发（在发现猎物的附近区域进行集中搜索）。全面的搜索可以发现猎物聚集的区域，为集中搜索提供搜索的区域，并能避免陷入猎物稀少的区域；集中的搜索可以在猎物集中的区域仔细地搜索猎物，防止漏掉猎物。由于捕食者大部分时间用在猎物聚集区，而猎物聚集区相对于整个搜索空间更容易发现猎物，所以动物的这种捕食策略是很高效的。关于动物捕食搜索策略的详细说明可以参见文献［1，3，4，5，10，12］。

8.2 基本原理

这里以 TSP 问题这一典型的组合优化问题为例来说明捕食搜索算法的实现。

8.2.1 捕食搜索算法的基本思想

模拟动物的这种捕食过程中的搜索策略，Alexandre Linhares 提出了一种新的针对组合优化问题的仿生算法，即捕食搜索（PS）算法，如图 8.2 所示。应用捕食搜索算法寻优时，先在整个搜索空间进行全局搜索，直到找到一个较优解；然后在较优解附近的区域进行集中搜索，直到搜索很多次也没有找到更优解，从而放弃局域搜索；然后再在整个搜索空间进行全局搜索。如此循环，直到找到最优解（或近似最优解）为止。在捕食搜索算法中，使用限制（Restriction）来表征较优解的邻域大小。通过限制的调节，实现搜索空间的增大和减小，从而达到探索能力和开发能力的平衡。

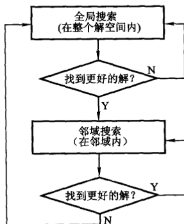


图 8.2 捕食搜索算法

捕食搜索算法不是一种具体的寻优计算方法，并没有给出在局域和全局如何进行具体的搜索，其本质上是一种平衡局域搜索和全局搜索的策略，所以可将其称为捕食搜索策略。局域搜索和全局搜索，广度探索和深度开发，搜索速度和优化质量是困扰着所有算法的矛盾。而捕食搜索非常巧妙地平衡了这个矛盾。捕食搜索在较差的区域进行全局搜索以找到较好的区域，然后在较好的区域进行集中地局域搜索，以使解得到

迅速地改善；捕食搜索的全局搜索负责对解空间进行广度探索，捕食搜索的局域搜索负责对较好区域进行深度开发；捕食搜索的局域搜索由于只集中在一个相对很小的区域进行搜索，所以搜索速度很快，捕食搜索的全局搜索可以提高搜索的质量，使搜索避免陷入局部最优点。

8.2.2 算法的实现

下面以 TSP 问题为例，说明算法的实现。

1. 算法中的基本概念

把组合优化问题定义为一个二元组（Ω, Z），这里 Ω 是解的集合，函数 Z: $\Omega \to \mathbf{R}$ 代表每个解到对应的适应值的变换。假设每个解 s 存在一个邻域 $N(s) \subset \Omega$，定义 $N'(s) \subset N(s)$，这里 $N'(s)$ 包含了 $N(s)$ 中元素的 5%。$N(s)$ 中一个解到另一个解的变换称为一个移动。

在规模为 n 个城市的 TSP 问题中，用由从 1 到 n 的 n 个自然数组成的集合中的每个元素表示城市，元素所在的位置表示被访问的顺序。例如，状态 s = （x_1, x_2, …, x_n）表示旅行商访问的路线为

$$x_1 — x_2 — \cdots — x_n — x_1$$

一个解为一个巡回路线，给定一个解 s，采用 2 - opt 法进行状态的转移，即将其子串

$$x_q,\ x_{q+1},\ \cdots,\ x_{q+r}$$

倒转得到

$$x_{q+r},\ \cdots,\ x_{q+1},\ x_q$$

则得到一个新的解，其中：$1 \leqslant q$，$q + r \leqslant n$。组合所有可能的 q 和 r 就得到 s 的邻域 $N(s)$。

如果对于任何两个状态 s_0 和 s_m，和某个 $R \in \mathbf{R}$，存在一个序列

$$s_0,\ s_1,\ s_2,\ \cdots,\ s_{m-1},\ s_m$$

若对于所有正整数 $0 \leqslant k < m$，有 $s_{k+1} \in N(s_k)$，则称解 s_m 是从 s_0 可达的。

如果对某个 $R \in \mathbf{R}$，对于路径上的所有状态有 $Z(s_k) \leqslant R$，则称这条路径服从限制（Restriction）R。因此可以将函数 A: $\Omega \times \mathbf{R} \to 2^{\Omega}$ 定义为从 s 可达的服从限制 R 的所有解的集合 $A(s, \mathbf{R}) \subset \Omega$。这样，给定一个最好解 b 和一个限制 R，围绕 b 的一个受限搜索区域可以表示为 $A(b, R) \subset \Omega$。为了实现一个在已知最好解附近的逐渐扩大的搜索区域，可以定义一个由 $NumLevel + 1$ 个限制级别组成的序列：限制 $[L]$，这里 $L \in \{0, 1, \cdots, NumLevel\}$，称为限制的级别（Level of the Restriction）。

图 8.3 给出了一个解空间的示意图，在已经找到的最优解的附近，使用限制来区分不同大小的搜索空间。图 8.3 中，解被表示为标有数字的圆，在 Z 坐标上的投影为其相应的适应值。需要指出的是算法不能把这张图在记忆中保存，只能够根据当前解运行并动态计算其可能的邻域。在图 8.3 中，解 1 为全局最优解，但是算法中并没有限制从图 8.3 中某一个解的邻域里发现具有更好适应值的解 x。图 8.3 中也给出了搜索区域中的一些限制：在限制［3］的范围内，解 1 拥有一条到达解 12 的路径，而在限制［2］的范围内解则不能到达解 12。对于解 1 来说，在不同的限制下相应的邻域空间中解的集合为

限制［0］：{1}；

限制［1］：{1，2}；

限制［2］：{1，2，4，6}；

限制［3］：图中的所有解。

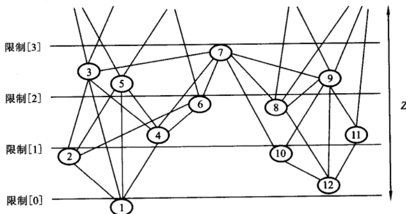


图 8.3 解空间示意图

算法的主要迭代来自搜索 1，从邻域中取样，如果样品中的最好解的适应值小于限制 L，那么将其作为新的解，重新开始。出于可操作性的考虑，算法的每步取样取一个较小的子集，即邻域的 5%。没有将整个集合 $N(s)$ 作为考察样本的原因：①这是一个耗时为 $O(n^2)$ 的操作；②将使算法成为不具可行性的操作，导致无限的循环。需要注意的是，邻域中的最好解即使其适应值大于当前解也会被接受，当移动出一个局优之后，算法必然还会返回该点，因为该点是邻域 $N(s)$ 中的最好解。

每一次尝试性的移动之后，有一个指针来记录在该区域内迭代的次数。当指针到达一个关键点之后，增加限制的等级 L，从而不断加大搜索的区域。当 L 到达某一个值 *Lthreshold* 时，意味着算法已经在所限制

的区域内进行了多次有效的搜索而没有找到改进的解（Improving Solution），于是算法放弃区域限制的搜索方式，将 L 设置为一个较高的值 *LhighThreshold*。这个较高的值表示在建立一列取样时得到的一个较差解的适应值，在这样一个限制的约束下，算法可以搜索很大的区域，很快跳出原来所限制的较小的区域。

当发现一个改进的解时，需要一些特殊的操作。

（1）若找到更好的解则更新最优解 b。

（2）从 b 的邻域中计算得到一列限制等级（具体后述）。

（3）将 L 设为 0，从而让算法在 b 的附近进行细致的搜索。

这也正是区域限制的搜索方式被触发的时刻，这里的捕食搜索策略的实现可以看做是捕食改进的解（Improving Solution），而不是捕食很好的解（Very Good Solution）。另一个可行的触发区域搜索的方式可以是在常规搜索中发现较好的解（即使其不优于当前的最好解），这样的触发条件将弱于前述方式。

2. 算法步骤

第 1 步：随机选择一个初始点 *solution*，$solution \in \Omega$，令 $b = solution$，$counter = 0$，$L = 0$。

第 2 步：如果 $L < NumLevel$，取 $N(s)$ 的 5% 构造出 $N'(s)$，并取其中最小解 *proposal*，然后转第 3 步；否则结束。

第 3 步：如果 $proposal \in A(b, Restriction\ [L])$，令 $solution = proposal$，然后转第 4 步；否则转第 5 步。

第 4 步：如果 $Z(solution) < Z(b)$，令 $b = solution$，$level = 0$，$counter = 0$，重新计算限制，然后转第 2 步；否则转第 5 步。

第 5 步：令 $counter = counter + 1$，如果 $counter > Cthreshold$，令 $L = L + 1$，$counter = 0$，然后转第 6 步；否则转第 2 步。

第 6 步：如果 $L = Lthreshold$，令 $L = LhighThreshold$，并转第 2 步；否则直接转第 2 步。

这里，*Restriction* [L] 表示限制，其他相关符号的定义，见算法概念中的描述。

3. 限制的计算

上述第 4 步中，如果 $Z(solution) < Z(b)$，重新计算限制，具体操作如下：

（1）搜索 *NumLevel* 次迄今为止发现的最好解 b 的邻域，计算 Z，得到 *NumLevel* 个适应值。

（2）把这 *NumLevel* 个值与发现的最好解的适应值按照升序排列。

(3) 把排列后的 *NumLevel* 个值依次赋给限制 *Restriction* [1], *Restriction* [2], …, *Restriction* [*NumLevel*]。而 *Restriction* [0] 的值取为刚获得的最好的适应值 $Z(b)$。

4. 参数的设置

在上述的算法实现中，文献 [2] 已经设置了一些特殊的参数，这里做以下说明。

(1) 每一步搜索时评估邻域中解的比例。邻域中可能的解数量为 $0.5(n^2-n)$，这里是取邻域中解的5%进行评估。

(2) *NumLevel*：总的限制级别数为 *NumLevel* + 1，文献 [2] 中 *NumLevel* 取为 n。

(3) *Cthreshold*：用于增大限制等级 L 的指针阈值。文献 [2] 中设置为 $3n$。如果在某个限制等级下，搜索了 *Cthreshold* 次（搜索一次即取样 5%）仍没有找到改进的解，那么将增大限制等级 L。

(4) *Lthreshold*：当 L 到达 *Lthreshold* 时，意味着算法已经在所限制的区域内进行了多次有效的搜索而没有找到改进的解，于是算法放弃区域限制的搜索方式，在文献 [2] 中将其设置为 $[n/20]$，这里 [ ] 为取整运算符，即区域限制搜索模式的限制等级数为城市总数被 20 除的结果的整数。

(5) *LhighThreshold*：表示一个较高的适应值，文献 [2] 中取为 $n-[n/20]$。即在常规搜索模式下，如果算法在 $[n/20]$ 个限制等级下搜索过后仍不能发现新的改进的解，那么算法将停止。

从上面的参数设置可以看到，在文献 [2] 中区域限制的搜索和常规搜索中经历的限制等级数都为 $[n/20]$，具体如下：

对于区域限制搜索来说，限制等级为

$$0, 1, 2, \cdots, [n/20]-1;$$

对于常规限制搜索来说，限制等级为

$$n-[n/20], n-[n/20]+1, n-[n/20]+2, \cdots, n-1$$

8.2.3 捕食搜索算法的应用条件

无可否认，只有当猎物聚集时，捕食者采用这种策略效率才高。可以想象，当猎物分布很扩散而且没有规律时，捕食者进行局域搜索只能无功而归。同样，捕食搜索也只有在较好解聚集时才有效。特别当较好解聚集于全局最优点时，捕食搜索的搜索效率很高。

UCLA 的 Boese K D，Kahng A B 和 Muddu S 利用贪婪算法（Greedy Descent Algorithm）求解了 100 个城市 Euclidean 旅行商问题的2 500个局

部最优解和 G（150，0.5）的二分图问题的2 500个局部最优解，并分别分析了这两个问题的2 500个局部最优解之间的关系（它们的各自的巡游路线长度和它们之间的距离的关系），得出结论：旅行商问题和二分图问题的较好的局部最优解聚集于全局最优解。

此后，Coventry 大学的 Reeves C R 求解了 $20/5/F/c_{max}$ 的流水车间问题的 50 个局部最优解，然后分析了它们的函数值与它们和已知的全局最优解的距离之间的关系，并得出了相同的结论。以上详细说明见参考文献［2，11］。

由此可以得出结论：捕食搜索是适合于求解旅行商问题（Traveling Saleman Problem，TSP），二分图问题（Graph Bisection）和流水车间问题（Flow - shop Sequencing）等组合优化问题的。

8.2.4 计算举例

五城市非对称 TSP 问题，设五个城市分别为：1，2，3，4，5，距离矩阵为

$$D=(d_{ij})=\begin{bmatrix}0&8&5&6&7\\6&0&8&5&9\\7&9&0&5&6\\9&7&8&0&5\\5&6&7&8&0\end{bmatrix}$$

为简化计算该问题的过程，相关参数设置如下：

（1）每一步搜索时评估邻域中解的数量为 2。

（2）$NumLevel=5$。

（3）$Cthreshold=1$。

（4）$Lthreshold=1$。

（5）$LhighThreshold=4$。

邻域定义如 8.2.2 节所示。

算法开始，随机取一个初始点：

$solution=(1, 2, 3, 4, 5)$

则 $b=solution$，$counter=0$，$L=0$。初始点的适应值为

$Z(b)=Z(solution)=8+8+5+5=26$

然后来计算限制，随机取 $solution$ 邻域中的五个解为

$s_1=(2, 1, 3, 4, 5)$

$s_2=(1, 2, 4, 3, 5)$

$s_3=(3, 2, 1, 4, 5)$

$s_4=(1, 2, 5, 4, 3)$

$s_5=(1, 5, 4, 3, 2)$

计算其相应的适应值为

$Z(s_1)=6+5+5+5=21$

$Z(s_2)=8+5+8+6=27$

$Z(s_3)=9+6+6+5=26$

$Z(s_4)=8+9+8+8=33$

$Z(s_5)=7+8+8+9=32$

所以，得到相应的限制为

Restriction [0] =26

Restriction [1] =21

Restriction [2] =26

Restriction [3] =27

Restriction [4] =32

Restriction [5] =33

至此，得到了第一列限制，邻域搜索的准备工作完毕。下面是捕食搜索的过程，从最小限制 *Restriction* [0] 开始。

(1) 第一列限制中的 *Restriction* [0] 限制下，*solution* 的邻域进行第一次搜索。

随机取 *solution* 的邻域中的两个解为

(2, 1, 3, 4, 5)，其适应值为 21

(1, 3, 2, 4, 5)，其适应值为 24

故 $proposal=(2, 1, 3, 4, 5)$，又 $Z(proposal)=21<Restriction[0]=26$

所以，$solution=(2, 1, 3, 4, 5)$，令 $b=solution$，重新计算限制。

随机取 *solution* 邻域中的五个解为

$s_1'=(1, 2, 3, 4, 5)$

$s_2'=(2, 1, 4, 3, 5)$

$s_3'=(3, 1, 2, 4, 5)$

$s_4'=(2, 1, 5, 4, 3)$

$s_5'=(2, 5, 4, 3, 1)$

计算其相应的适应值为

$Z(s_1')=8+7+6+5=26$

$Z(s_2')=6+6+8+6=26$

$Z(s_3')=7+8+5+5=25$

$Z(s_4')=6+7+8+8=29$

$Z(s_5')=9+8+8+7=32$

所以，得到相应的限制为

Restriction [0] =21

Restriction [1] =24

Restriction [2] =25

Restriction [3] =26

Restriction [4] =29

Restriction [5] =32

至此得到了第二列限制。

(2) 第二列限制中的 *Restriction* [0] 限制下，*solution* 的邻域进行第一次搜索。

随机取 *solution* 的邻域中的两个解为

(2, 1, 4, 3, 5)，其适应值为 24

(2, 1, 3, 5, 4)，其适应值为 25

故 *proposal* = (2, 1, 4, 3, 5)，又 *Z* (*proposal*) = 24 > *Restriction* [0] =21，不更新 *solution*，*counter* = *counter* + 1 = 1。

(3) 第二列限制中的 *Restriction* [0] 限制下，*solution* 的邻域进行第二次搜索。

再次搜索 *solution* = (2, 1, 3, 4, 5) 的邻域。随机取 *solution* 的邻域中的两个解为

(3, 1, 2, 4, 5)，其适应值为 25

(2, 4, 3, 1, 5)，其适应值为 27

所以 *proposal* = (3, 1, 2, 4, 5)

由 *Z*(*proposal*) = 25 > *Restriction* [0] = 21，于是不更新 *solution*，*counter* = *counter* + 1 = 2 > *Cthreshold* = 1，到达了指针阈值，所以增大限制等级，*L* = *L* + 1 = 1，计数器清零 (*counter* = 0)。

(4) 第二列限制中的 *Restriction* [1] 限制下，*solution* 的邻域进行第一次搜索。

在 *solution* 的邻域内随机取两个解为

(2, 1, 5, 4, 3)，其适应值为 29

(4, 3, 1, 2, 5)，其适应值为 32

所以 *proposal* = (2, 1, 5, 4, 3)，由 *Z*(*proposal*) = 29 > *Restriction* [1] = 24，不更新 *solution*，*counter* = *counter* + 1 = 1。

(5) 第二列限制中的 *Restriction* [1] 限制下，*solution* 的邻域进行第二次搜索。

在 *solution* 的邻域内随机取两个解为

(2, 5, 4, 3, 1), 其适应值为 32

(5, 4, 3, 1, 2), 其适应值为 31

所以 *proposal* = (5, 4, 3, 1, 2), 由于 $Z(proposal) = 31 > Restriction[1] = 24$, 不更新 *solution*, $counter = counter + 1 = 2 > 1$, 所以增大限制等级并且令 $counter = 0$, $L = L + 1 = 2 > Lthreshold = 1$, 所以结束邻域限制搜索, 令 $L = LhighThreshold = 4$, 下面将开始常规搜索, 即广域搜索。

(6) 第二列限制中的 *Restriction* [4] 限制下, *solution* 的邻域进行第一次搜索。

取 *solution* 邻域中的两个解为

(3, 1, 2, 4, 5), 其适应值为 25

(5, 4, 3, 1, 2), 其适应值为 31

proposal = (3, 1, 2, 4, 5), $Z(proposal) = 25 < Restriction[4] = 29$, 所以 *solution* = *proposal*, 由于 $Z(solution) = 25 > Z(b) = 21$, 所以不更新最优解 b。

令 $counter = counter + 1 = 1$, 再次进行局域搜索, 即第二列限制中的 *Restriction* [4] 限制下, *solution* 的邻域进行第二次搜索。

从上面的计算过程, 可以看到:

①算法初始阶段, 由于初始解随机选取, 导致 *Restriction* [0] 可能大于 *Restriction* [1], 但是邻域选优的结果将使 *Restriction* [0] 快速下降。

②在第6步中, 即第二列限制中的 *Restriction* [4] 限制下, *solution* 的邻域进行第一次搜索后, 得到的 *proposal* 虽然没有优于已知最优解, 但是由于其小于当前的限制, 所以当前的解 *solution* 仍然移动到 *proposal* 处, 这将使捕食搜索具有跳出局优的能力。

8.3 改进与变形

通过设置限制缩小搜索空间, 提高搜索效率, 是捕食搜索的精华之所在。所以, 限制方法的选取也就成为影响捕食搜索算法搜索效率的一个关键因素。如果限制太宽, 搜索区域变大, 将花费很多时间才能找到较好解; 如果限制太窄, 较好解可能不在搜索区域里, 而导致解得不到改善。Linhares A 在其初始的捕食搜索算法中, 使用了解的函数值 (Restricted by Solution Cost) 作为限制, 来实现搜索空间大小

的变化。

刘崇等分析认为，用函数值作为限制有时可能导致搜索的效率低和搜索的盲目性，并提出了使用距离限制的捕食搜索算法，即利用与迄今为止发现的最好解之间的距离对算法的搜索空间进行限制。并且将改进算法应用于解决 TSP 问题和流水车间调度问题。这里同样以 TSP 问题为例介绍使用距离限制的捕食搜索算法。

8.3.1 TSP 巡游路线之间的距离

TSP 巡游路线之间的距离根据不同的定义有两种表示方法。

(1) 2 - opt 距离 (2 - opt Distance)

2 - opt 距离 $d(t_1, t_2)$ 定义为从 t_1 通过 2 - opt 邻域变换变为 t_2 的最小变换次数。

(2) Bond 距离 (Bond Distance)

Bond 距离 $b(t_1, t_2)$ 定义为用城市数 n 减去 t_1 与 t_2 相同的边数。

这里采用第二种距离定义，即 Bond 距离。

8.3.2 算法步骤

第 1 步：随机选择一个初始点 *solution*，$solution \in \Omega$，令 $b = solution$，$counter = 0$，$L = 0$。

第 2 步：如果 $L < NumLevel$，取 $N(s)$ 的 10% 构造出 $N'(s)$，并取其中最小解 *proposal*，然后转第 3 步；否则结束。

第 3 步：计算 *proposal* 与 b 之间的 Bond 距离。

第 4 步：如果 $Bond(proposal, b) \leqslant Restriction[L]$，令 $solution = proposal$，然后转第 5 步；否则转第 6 步。

第 5 步：如果 $Z(solution) < Z(b)$，令 $b = solution$，$level = 0$，$counter = 0$，重新计算限制，然后转第 2 步；否则转第 6 步。

第 6 步：令 $counter = counter + 1$，如果 $counter > Cthreshold$，令 $L = L + 1$，$counter = 0$，然后转第 7 步；否则转第 2 步。

第 7 步：如果 $L = Lthreshold$，令 $L = LhighThreshold$，并转第 2 步；否则直接转第 2 步。

8.3.3 限制的计算

该算法的限制条件是固定的值，不随历史最优解的变化而变化。也可以取一组升序的数组作为限制，对算法的搜索区域进行分级限制。由于解之间的距离本身就是相对于当前最好解的距离，所以当发现新的最

好解时，也不用重新计算限制。以下操作只在算法初始化时执行。

①把2，12，…，$[n/2]-8$赋给 $Restriction[0]$，$Restriction[1]$，…，$Restriction[[n/20]-1]$；

②把 $[n/2]$ 赋给其余的限制。

8.3.4 参数的设置

文献［4］对上述算法中的参数设置如下。

①每一步搜索时评估邻域中解的比例。即邻域搜索的数目，设为整个邻域的10%，为 $0.05n(n-1)$。

②$NumLevel$：也取为 n。

③$Cthreshold$：也设置为 $3n$。

④$Lthreshold$：也设置为 $[n/20]$。即对于区域搜索，限制的等级从0到 $[n/20-1]$，等差为1。

⑤$LhighThreshold$：文献［4］中取为 $n-[n/40]$。对于全局搜索，限制的等级从 $[n-n/40]$ 到 $n-1$，等差也为1。

注意算法的参数设置，不难发现，在全局搜索过程中，变形算法的限制水平的级数是局域搜索中限制水平级数的一半。这样的设置意味着，以Bond距离为限制的捕食搜索算法更注重了局域搜索。

8.4 应用实例

8.4.1 电子商务中物流配送路径优化的问题描述与模型

B2C电子商务中物流配送路径优化模型的基本思想可描述如下（本例源于文献［13］）：根据B2C电子商务企业在某个时段内顾客的订货情况（如顾客商品需求量和其地理位置），利用信息技术（如GIS技术）确定该时段的实际配送网络，通过优化设计一套基于配送网络的车辆路径，同时要满足一系列的约束条件（如商品需求量、配送中心和车辆容量限制等），使得配送总费用最小。这里总费用包括车辆配送费用和车辆一次性启动费用。为了便于建立模型，将实际物流配送网络用由配送中心和顾客两类节点构成的不完全无向图表示，并作以下几个基本假设：

①配送中心有多个，每个配送中心各类商品量以及配送车辆数一定。每辆车仅隶属于一个配送中心。

②每个顾客仅能由一个配送中心中的一辆车进行一次性商品配送，

但可以被多次访问。特殊地，如果顾客需求超出一辆车的容量则选择最近可用的配送中心由多辆车对其进行配送，因而此类情况在通过数据预处理后亦可由模型表示。

③每辆车从各自的配送中心出发，完成配送任务后返回自己所在的配送中心。

④配送商品为多品种商品，配送车辆为单一类型车辆。

下面给出 B2C 电子商务中物流配送路径优化的数学模型：

$$Min \sum_{k \in K} \sum_{\substack{i \in H \\ (i,j) \in E}} \sum_{\substack{j \in H \\ (i,j) \in E}} C_{ij} d_{ij} x_{ijk} + \sum_{k \in K} B_k \underset{j \in D}{Max}(z_{jk}) \tag{8.1}$$

$$\text{s.t.} \sum_{\substack{i \in D \\ (i,p) \in E}} x_{ipk} - \sum_{\substack{j \in D \\ (p,j) \in E}} x_{pjk} = 0, \forall p \in S, k \in K_p \tag{8.2}$$

$$\sum_{\substack{i \in H \\ (i,j) \in E}} x_{ijk} \geq z_{jk}, \forall j \in D, k \in K \tag{8.3}$$

$$\sum_{k \in K} z_{jk} = 1, \forall j \in D \tag{8.4}$$

$$\sum_{k \in K_p} \underset{j \in D}{Max}(z_{jk}) \leq A_p, \forall p \in S \tag{8.5}$$

$$\sum_{j \in D} \left(\sum_{l \in L} w_l q_{jl} \right) z_{jk} \leq Q, \forall k \in K \tag{8.6}$$

$$\sum_{j \in D} q_{jl} y_{ij} - V_{il} \leq 0, \forall i \in S, l \in L \tag{8.7}$$

$$x_{ijk} = 0,1 \quad \forall i \in H, j \in H, k \in K \tag{8.8}$$

$$y_{ij} = 0,1 \quad \forall i \in S, j \in D \tag{8.9}$$

$$z_{jk} = 0,1 \quad \forall j \in D, k \in K \tag{8.10}$$

模型中符号有两类，即模型的决策变量和模型参数。

(1) 决策变量

x_{ijk}——表示车辆 k 是否从顾客（或配送中心）i 开往 j（并不一定给顾客 j 配送商品），如果是，其值为 **1**，否则为 **0**。

y_{ij}——表示顾客 j 是否由配送中心 i 负责配送，如果是，其值为 **1**，否则为 **0**。

z_{jk}——表示顾客 j 是否由车辆 k 配送，如果是，其值为 **1**，否则为 **0**。

(2) 模型参数

G——配送中心、顾客两类节点和代表它们之间配送线路的边组成的不完全无向图，$G = \{S, D, E\}$，其中：$S = \{n+1, \cdots, n+m\}$ 表示配送中心的节点集合，$D = \{1, \cdots, n\}$ 表示顾客的节点集合，$E = \{(i, j), i, j \in H = S \cup D\}$ 表示配送中心、顾客间直接线路的边的集合。

K——配送车辆集合 $\{K_{n+1}, \cdots, K_{n+m}\}$，其中，$K_i$表示配送中心 i 车辆的集合，$K_i \cap K_j = \varnothing$，$(i \neq j)$，即每辆车仅隶属于一个配送中心。

L——配送商品种类集合 $\{1, \cdots, T\}$。

A_p——配送中心 p 的可用车辆数。

B_k——车辆 k 的一次性起动费用（主要考虑车辆的占用和损耗费用，略去停留费用）。

C_{ij}——车辆 k 在路线 (i, j) 单位路程的运输费用（忽略车辆装载量大小对运输费用的影响）。

q_{jl}——顾客 j 对商品 l 的需求量。

d_{ij}——顾客（或配送中心）i，j 间的距离，即线路 (i, j) 的路程。

Q——单个车辆的装载量。

w_l——商品 l 的重量（或体积）系数。

V_{il}——配送中心 i 商品 l 的供应量。

目标函数式（8.1）由两部分组成，第一部分是车辆的配送费用，第二部分是车辆启动的一次性费用，这里是根据使用车辆数进行计算。约束式（8.2）保证从所属配送中心出发的车辆返回到该配送中心；约束式（8.3）表示顾客 j 如果由车辆 k 配送商品，则车辆 k 至少访问顾客 j 一次；约束式（8.4）表示每个顾客仅由一辆车配送；约束式（8.5）表示每个配送中心可用车辆数限制；约束式（8.6）保证每辆车装载量不超过其容量；约束式（8.7）表示每个配送中心各类配送商品的供应量；式（8.8）、式（8.9）和式（8.10）分别为对应的 **0-1** 决策变量。

上述模型为一种改进的多设施 VRP 模型，其改进之处在于结合了 B2C 电子商务企业实际配送网络，模型中的决策变量 x_{ijk} 可以直接表示出基于配送网络的车辆路径，因而比通常的多设施 VRP 模型更贴近现实情况。

8.4.2　模型求解的捕食搜索算法

下面使用捕食搜索算法来对该模型进行求解。

1. 解的表达

采用顺序编码，将无向图中的 $m-1$ 个配送中心和 n 个顾客一起进行编码。例如，3 个配送中心，10 个顾客，则编码可为：1-2-3-4-0-5-6-7-0-8-9-10，其中 0 表示配送中心，上述编码表示配送中心 1 负责顾客 1，2，3，4 的配送，配送中心 2 负责顾客 5，6，7 的

配送，配送中心 3 负责顾客 8，9，10 的配送。然后对于每个配送中心根据顾客编码中的顺序进行车辆的分配，这里主要考虑车辆的容量约束。依此编码方案，随机产生初始解。

2. 邻域定义

采用逆转法实现邻域的操作，即随机选择解的两个位置将它们之间的编码进行逆转得到当前解的一个邻域。

3. 目标值的确定

目标值 $f(x)$ 由编码解码得到，对于没有直接最短线路的顾客间配送费用由 FLOYD 算法求得。另外对超出配送中心商品数和车辆数的解的目标值给予一定惩罚，惩罚与其超量成比例。

4. 算法步骤

（1）算法流程

第 1 步：随机产生一个初始解 x，令历史最好解 $x_{min}=x$，限制级别 $level=0$，循环次数 $counter=0$。

第 2 步：如果 $level<n+m-1$，搜索 x 的邻域 s 次（s 可取问题的规模数 $n+m$），并取其最小解 x_{n_min}，然后转第 3 步，否则结束。

第 3 步：如果 $f(x_{n_min}) \leqslant Restriction(level)$，令 $x=x_{n_min}$，然后转第 4 步，否则转第 5 步。

第 4 步：如果 $f(x) < f(x_{min})$，令 $x_{min}=x$，$level=0$，$counter=0$，重新计算限制，然后转第 2 步，否则转第 5 步。

第 5 步：令 $counter=counter+1$，如果 $counter>COUNTER_MAX$（最大循环次数），令 $level=level+1$，$counter=0$，然后转第 6 步，否则转第 2 步。

第 6 步：如果 $level=[(n+m-1)/t]$，令 $level=n+m-1-[(n+m-1)/t]$（通过限制的跳跃实现从局域搜索到全局搜索的转换，[ ] 表示取整数），t 的值视 $n+m-1$ 的大小而取值，并转第 2 步，否则直接转第 2 步。

（2）限制（*Restriction*）的计算

每当解得到改善时（即获得了一个当前最好的解），执行以下操作得到新的限制。

① 搜索 $n+m-2$ 次迄今为止发现的最好解的邻域，得到 $n+m-2$ 个解的值。

② 把这 $n+m-2$ 个值与发现的最好解的值按照升序排列。

③ 把排列后的 $n+m-1$ 个值依次赋给限制 $Restriction[0]$，…，$Restriction[n+m-2]$。

8.4.3 仿真结果与比较分析

设某B2C电子商务企业在某时段由3个配送中心为17个顾客配送3类商品，配送网络如图8.4所示。为计算简捷，设各配送中心可用车辆数 $A_p=3$ 辆，最大载重量 $Q=10$ t，车辆启动费用 $B_k=400$ 元，单位距离费用 $C_{ij}=5$ 元，3类商品的重量系数分别为 $w_1=0.2$ 吨/件、$w_2=0.4$ 吨/件、$w_3=0.3$ 吨/件，其他相关参数见表8.1。

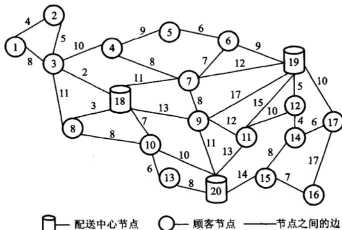


图8.4 实际配送网络图

表8.1 顾客对3类商品的需求量和配送中心供应量（单位：件）

顾客	需求量	顾客	需求量	顾客	需求量
1	(2, 1, 3)	8	(1, 2, 3)	15	(1, 1, 1)
2	(3, 2, 1)	9	(2, 1, 1)	16	(3, 2, 1)
3	(4, 1, 2)	10	(1, 2, 1)	17	(1, 1, 2)
4	(1, 1, 1)	11	(2, 3, 2)	配送中心	商品量
5	(3, 3, 2)	12	(1, 1, 1)	18	(31, 32, 28)
6	(1, 2, 3)	13	(4, 2, 2)	19	(29, 37, 30)
7	(2, 1, 2)	14	(2, 1, 2)	20	(33, 35, 32)

捕食搜索算法采用Java语言在Windows平台上实现。其中邻域搜索次数 $s=20$，$COUNTER_MAX=80$，$t=6$，惩罚系数为500。求得最优解值和车辆配送路径见表8.2，可以看出结果能直接得到基于配送网络的车辆实际配送路径，而且有些节点被多次访问（如节点3、10），该类顾客仅

在第一次被访问的时候对其进行配送服务（表 8.2 配送路径中黑体节点表示车辆配送的顾客和其次序），这和实际配送情况亦是相符的。

表 8.2 最优计算结果及最优配送路径

	车辆和其相应的配送顾客	最优配送路径	配送费用/元	车辆启动费用/元	总费用（最优目标值）/元
配送中心 1	8,10,13,1,2	18 - **8** - **10** - **13** - 10 - 18 - 3 - **1** - **2** - 3 - 18	530	1 200	2 085
	3,4,5,6,7,9	18 - **3** - **4** - **5** - **6** - **7** - **9** - 18			
配送中心 2	17,16,15,14,12,11	19 - **17** - **16** - **15** - **14** - **12** - **11** - 19	355		
配送中心 3	—	—	0		

为了验证捕食搜索算法的有效性，将捕食搜索算法与基于类顺序交叉和换位变异算子的遗传算法（编码相同，交叉率为 0.8，变异率为 0.05，种群 80，迭代次数 500）对上述算例各随机计算 10 次，得到相应的目标值和计算时间如表 8.3 所示。

表 8.3 捕食搜索算法（PS）与遗传算法（GA）计算结果比较

计算次数	目标值/元		计算时间/秒	
	GA	PS	GA	PS
1	2 380.0	2 085.0	0.951	0.441
2	2 265.0	2 085.0	0.892	0.460
3	2 250.0	2 085.0	0.901	0.531
4	2 440.0	2 085.0	0.891	0.631
5	2 290.0	2 085.0	0.912	0.461
6	2 105.0	2 085.0	0.821	0.400
7	2 315.0	2 085.0	0.831	0.561
8	2 285.0	2 085.0	0.831	0.491
9	2 345.0	2 205.0	0.851	0.180
10	2 185.0	2 085.0	0.811	0.491
平均值	2 286.0	2 097.0	0.869	0.465
标准差	285.6	113.8	0.140	0.358

由表8.3中可以看出，捕食搜索算法求得的目标值优于遗传算法，10次计算中9次得到了最优值（或近似最优值）2 085.0元，而遗传算法的最好值仅为2 105.0元，计算的平均值，捕食搜索算法比遗传算法低8.28%，而且捕食搜索的计算标准差比遗传算法小，说明捕食搜索算法求得优化解的稳定性强于遗传算法。从计算的时间来看，捕食搜索算法的计算效率高于遗传算法，但是前者的计算时间没有后者稳定，可以从它们计算时间的标准差上看出这一点，这是因为遗传算法在算法参数设定后计算时间波动很小（以最大迭代次数为停止准则），而捕食搜索算法因为模仿动物捕食的内在特点，除了算法参数外，其初始解亦会影响算法的计算时间。上述两种算法在多个算例上进行了实验，得到了相似的结论。因而，文中设计的捕食搜索算法对模型的求解是可行和高效的。

图8.5给出了对应网络图中最优解的车辆配送路径。

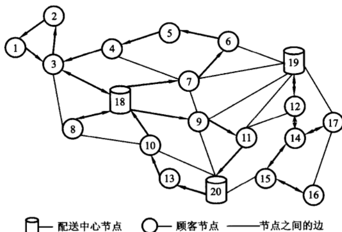


图8.5 车辆最优配送路径

问题与思考

1. 流水车间调度问题一般可以描述为 n 个工件要在 m 台机器上加工，每个工件需要经过 m 道工序，每道工序要求不同的机器。n 个工件在 m 台机器上的加工顺序相同。工件 i 在机器 j 上的加工时间是给定的，设为 t_{ij}（$i=1, \cdots, n$; $j=1, \cdots, m$）。问题的目标是求 n 个工件在每台机器上最优的加工顺序，使最大流程时间到达最小。对该问题常作如下假设：（1）每个工件在机器上的加工顺序是 $1, 2, \cdots, m$；（2）每台机器同时只能加工一个工件；（3）一个工件不能同时在不同机器上加工；（4）工序不能预定；（5）工序的准备时间与顺序无关，且包含在

加工时间中；(6) 工件在每台机器上的加工顺序相同，且是确定的。

试用捕食搜索算法求解上述问题。要求写明编码方式、邻域构成策略、限制的计算方式、停止准则，并画出流程图。

2. 试比较捕食算法中通过限制变化实现的外循环与模拟退火中通过温度变化实现的外循环有何异同。

3. 思考：能否将捕食搜索算法应用于解决连续优化问题。

4. 试为捕食搜索算法设计优化约束问题的处理方法。

参考文献

[1] Bell W J. Searching behavior patterns in insects [J]. Annu. Rev. Entomology, 1990 (35): 447 -467.

[2] Boese K D, Kahng A B, Muddu S. A new adaptive multi-start technique for combinatorial global optimizations [J]. Oper. Res. Lett., 1994, vol. 16: 101 -113.

[3] Curio E. The Ethology of Predation [M]. Berlin, Germany: Springer-Verlag, 1976.

[4] Huey R B, Pianka E R. Ecological consequences of foraging mode [J]. Ecology, 1981 (62): 991 -999.

[5] Kareiva P, Odell G. Swarms of predators exhibit 'preytaxis' if individual predators use area-restricted search [J]. Amer. Naturalist, 1987 (130): 233 -270.

[6] Linhares A. Preying on optima: A predatory search strategy for combinatorial problems: Proc. IEEE Int. Conf. Syst., Man, Cybern. [C]. 1998: 2974 -2978.

[7] Linhares A. State-space search strategies gleaned from animal behavior: A traveling salesman experiment [J]. Biological Cybernetics, 1998, 78: 167 -173.

[8] Linhares A. Synthesizing a Predatory Search Strategy for VLSI Layouts [J]. IEEE Trans. on Evolutionary Computation, 1999, 3 (2): 147 -152.

[9] Liu C, Wang D. Predatory search algorithm with restriction of solution distance [J]. Biological Cybernetics, 2005 (92): 293 -302.

[10] Nakamuta K. Mechanism of the switchover from extensive to area concentrated search behavior of the ladybird beetle, coccinella septempuc-

tata bruckii [J]. J. Insect Physiol., 1985 (31): 849 - 856.

[11] Reeves C R. Landscapes, operators, and heuristic search [J]. Ann. Oper. Res., submitted for publication, 1999 (86): 473 - 490.

[12] Smith J N M. The food searching behavior of two European thrushes. II. The adaptiveness of the search patterns [J]. Behavior, 1974 (59): 1 - 61.

[13] 蒋忠中，汪定伟. B2C电子商务中物流配送路径优化的模型与算法 [J]. 信息与控制，2005，34 (1)：481 - 485.

第9章　动态进化算法

将进化算法应用于动态环境的研究是当前进化计算的研究热点之一。由于传统的进化算法当其进化后期会失去对环境变化的适应能力，这是进化算法在动态环境中所面临的主要挑战。本章对动态环境的特征、标准的动态优化问题和性能评估方法以及怎样探测环境中的变化等作一介绍，最后介绍一种求解动态优化问题的原对偶遗传算法及其改进。

9.1　导　言

很多复杂的现实世界中的优化问题都是动态的，会随时发生变化。例如新的 Job 不断到达，需要立即加入到当前调度中；机器可能会发生故障或者降低加工速度；原材料的质量也会发生变化以及需要考虑生产限度的影响等。在解决这些动态优化问题的时候，需要一类功能强大的能够处理这些在现实世界中普遍存在的不确定性的启发式算法。

作为一种鲁棒的优化方法，进化算法（Evolutionary Algorithms，EAs）经过近几十年的发展一直被广泛的应用于各种静态优化问题中，而且在这个领域的研究至今仍旧保持着快速的发展。然而，EAs 也可以被认为特别适用于动态优化和不确定性优化问题，这主要是由于其基于自然选择和生物进化的机制实际上也是一种随机的动态过程。

静态优化问题的特点是在其优化计算过程中，问题本身或者外部环境并不会发生任何变化，最优解也始终保持不变。然而在动态环境中，问题的最优解会因为目标函数、环境参数或者约束条件的变化而随时产生变化。在处理这种动态情况时，一个最简单的方法就是将每一次变化的发生都作为一个新优化问题的开始，然后从头进行求解。但是这种简单的方法通常是不切实际的，其主要原因是：

①不重用过去的信息而从头求解问题会太浪费时间。

②环境的变化可能很难被发现，或者至少在一段时间内是未被发现的。

③新问题的解与旧问题的解不应该相差太多。

④每次小变化后都重新对问题进行求解是不可能的或者被认为是不

经济的。

这就需要这样一种优化算法，它不仅能够持续地适应环境的变化，而且能够重用到过去的信息。进化算法因其计算方法与自然界中的适应很类似，是一个不断持续的过程，看起来能够作为解决动态优化问题的不错选择。

将进化算法应用于动态环境的研究最早可以追溯到1966年，但是直到20世纪80年代中期才成为众多学者的研究热点之一。由于传统的进化算法的目标是使种群逐渐收敛，最终获得一个满意解，这种做法会使种群失去多样性，而种群的多样性恰恰是有效地探索整个可行空间的必要条件。因此传统的进化算法当其进化后期会失去对环境变化的适应能力，这是进化算法在动态环境中所面临的主要挑战。本章对动态环境的特征、标准的动态优化问题和性能评估方法以及怎样探测环境中的变化等作一介绍，最后介绍了 Yang 在 CEC2003（Congress on Evolutionary Computation 2003）上提出的一种求解动态优化问题的原对偶遗传算法及其改进。

9.2 动态环境的特征

一般来说，任何时变的问题都可以认为是动态的，然而按照进化算法的观点，并不是所有这样的问题都是值得感兴趣的。

首先，本章所关注的是那些函数适值曲线在变化前后表现出某种可利用的相似性的问题，如果该问题完全改变，与过去没有任何联系，那么它应该被认为是一个完全独立的问题，只能从头进行求解，因为没有任何信息能够通过持续的适应而获得。

其次，即使问题看上去存在着某种潜在的动态情况，还需要看算法是否去处理这些动态情况。比如，如果 EA 的任务是对一个给定的动态问题设计一个模糊控制器，那么这个 EA 优化问题仍旧被认为是静态的，因为这个最优控制器（也可以认为是 EA 所发现的最优解）并不是随时变化的。另一方面，同样在这个问题中如果 EA 直接被用于持续地优化这些控制变量，那么它将被认为是动态的。

此外，对于噪声适值函数的优化问题并不属于本章内容的研究范畴，尽管这类优化问题与动态优化问题很类似，然而却完全不同。因为尽管存在某种噪声，但优化的目标仍然只是为了寻找一个静态的最优解，而在动态环境中，最优解不是唯一的，是变化的。

尽管本章仅仅关注上面所描述的动态情况，但并不是所有的动态环

境都是一样的，不同的动态问题需要不同的优化方法来解决。本节接下来介绍了各种用于区分不同动态环境的特征。

(1) 变化频率 (Frequency of Change)：环境多长时间变化一次（从缓慢变化到频繁变化）？或者，更重要的是，EA 在多长时间内去获得适合的解？由于算法运行时间的比较依赖于硬件和具体的执行过程，而且通常在 EA 中估值次数是时间的主要决定要素，因此两次变化间的平均估值次数通常作为估量变化频率的合适指标。除非对于计算时间存在着其他的相关要求，否则环境两次变化间的实际时间一般不会作为衡量频率的指标。

(2) 变化强度 (Severity of Change)：系统变化得有多强烈？仅仅是一个微小的改变还是一个全新的情况？这主要取决于新旧最优解基因型之间的距离，当然也可能取决于其他的一些因素，比如搜索空间的大小，新最优解是否可以由旧最优解通过简单的爬坡获得，或者新最优解能否由旧最优解通过简单的变异获得。另外，在确定变化强度大小的时候，还可以进一步考虑搜索空间中每个点质量上的平均变化程度，对于高性能区域可能会给予更高的权重等。

(3) 变化可预测性 (Predictability of Change)：在变化中是否存在某种趋势或者模式，还是仅仅是随机的？根据当前面对的变化是否能够预测下一次变化的趋势、时间或者强度？

(4) 循环长度和准确性 (Cycle Length/Cycle Accuracy)：最优点是否会回到以前的位置或者至少很接近？如果是这样，怎样接近？循环长度就是用来评估环境回到相同状态或者很相似状态时经历的平均状态数的指标。如果新状态并不完全一样而是有所不同，那么新解与以前遇到状态的解之间的差异就变得很重要。循环长度和准确性可能决定记忆以前的解是否是一个有用的策略。

对于不同的动态环境，由于所表现出的各种动态特征可能不尽相同，所以一般来说很难对它们进行比较。但至少可以在同一个问题中对上述各种动态特征进行变化，以便于检验这些特征对一个特殊 EA 性能的影响。除了上面介绍的这些环境特征之外，在设计一种应用于动态环境中的 EA 时考虑如下 4 个方面的性质也是很令人感兴趣的。

(1) 变化的可预知性 (Visibility of Change)：变化的发生对于系统来说是否是已知的，还是需要去探测？

(2) 编码改变的必要性 (Necessity to Change Representation)：个体的基因型是否被变化所影响，比如说问题的维数发生了变化。

(3) 变化的形式 (Aspect of Change)：变化是属于目标函数的变

化、环境变量的变化还是某些约束的变化？

（4）EA对环境的影响：通常，EA产生的解会影响到环境，比如说在调度问题中，产生的解在环境变化之前可能已经被部分完成了，这就对EA的将来行为的选择产生限制，或者在并行进化模块中，一个种群的进化可能会影响到其他种群的进化。

此外，Weicker还提出了另一类关于动态环境特征的分类方法。在这种方法中，环境可以被划分为持续变化型（每个周期都发生相同的变化），静态型（不变化），周期型（回到以前的状态），一致型（整个适值曲线各个部分的运动是一致的，并非不同部分变化也不同）和跳跃型（最优解可以从适值曲线的一个部分或者峰上随机的跳跃到另一部分或者另一个峰上）等。

9.3 动态测试问题

迄今为止，有超过20多种动态优化问题被用于检验各种算法的性能，从简单的数学函数到各种调度问题乃至动态环境中各种人工生命方法的应用等。然而大多数问题都是特殊的，并不能引起大多数研究者的共同兴趣。为了更好地比较动态环境中不同算法的性能，本节接下来将介绍一组标准的动态测试函数。这些函数具有如下共同性质。

①能够变化9.2节中定义的各种环境变量。

②能够使用二进制编码或者实数编码。

③能够容易地描述函数曲线。

④易于分析。

⑤计算效率高。

⑥能够与现实问题相联系。

9.3.1 动态位匹配问题

静态位匹配问题是一类经常用于检验各种算法性能的测试函数，通常将个体与给定模板相匹配的位数作为个体的适值。如果模板会随时变化，那么这个问题就变成了动态位匹配问题（Dynamic Bit-Matching）。这类问题相对容易定义、容易分析而且特别适合二进制编码方式。变化的频率可以直接利用评估次数来设定，而强度可以通过每次变化时给定模板的变化位数所决定，也可以通过增加的串长所决定。对于其他特征比如说循环的可预测性和长度可以通过直接重复一个模块序列来实现。

这类问题的缺陷主要是函数为单峰函数，有时候一个简单的爬坡算

法可能会比 EA 更适用，而且通常被限制为二进制编码。

9.3.2 移动抛物线

移动抛物线（Moving Parabola）问题是与位匹配函数最为类似的一类实编码的动态问题。与动态位匹配问题一样，这类问题容易检验，便于定义和分析，可以任意变动维数，而且同样具有单峰函数的局限。函数的变化可以通过抛物线的移动来表现，其强度可以通过移动的幅度来控制。另外还可以通过某种特定的函数来决定抛物线的运动方向、循环长度和准确性。

Angeline 研究了三种不同动态模式（线性、循环和随机）下的移动抛物线函数。设 s 为一个设定变化强度的参数，n 为维数，$N(0, 1)$ 表示一个正态分布的随机变量，则这种函数能够通过式 9.1 来计算。另外，t 被用来记录环境变化的次数，环境发生一次变化（也就是每经过 Δe 次估值），t 就加 1。

$$f(x,t) = \sum_{i=1}^{n} [x_i + \delta_i(t)]^2 \tag{9.1}$$

线性变化

$$\delta_i(0) = 0 \quad \forall i \in \{1, \cdots, n\}$$
$$\delta_i(t) = \delta_i(t-1) + s$$

随机变化

$$\delta_i(0) = 0 \quad \forall i \in \{1, \cdots, n\}$$
$$\delta_i(t) = \delta_i(t-1) + s \cdot N_i(0,1)$$

循环变化

$$\delta_i(0) = \begin{cases} 0, i\text{ 为奇数} \\ s, i\text{ 为偶数} \end{cases}$$

$$\delta_i(t) = \delta_i(t-1) + s \cdot c(i,t)\text{，其中，}c(i,t) = \begin{cases} \sin\left(\dfrac{2\pi t}{\gamma}\right), i\text{ 为奇数} \\ \cos\left(\dfrac{2\pi t}{\gamma}\right), i\text{ 为偶数} \end{cases}$$

其中，γ 表示环境能够经历的不同状态数，由此可见 γ 能够决定环境的变化周期。

9.3.3 时变背包问题

时变背包问题（Time - Varying Knapsack Problem）是迄今为止最为常用的一类动态测试问题，其动态性主要体现在允许背包的重量限制变

化。这类问题存在着一种特殊的性质，也就是说，当重量限制降低时，旧解通常变成了无效解。此时，基本上需要丢弃旧解，促使 EA 寻找新解。

在大多数论文中，都是让重量限制仅仅在两个值之间变化，也就是说，环境仅仅在两种状态下振荡。当然在这种情况下，记忆将会很有用。当然也有一些研究者讨论了重量限制在三个值之间变化的时变背包问题，其变化可以是随机的，也可以是定期逐步下降的。

9.3.4 移动峰函数

为了有效地衔接复杂的不易于理解的现实世界中的优化问题与简单的动态测试问题，Branke 提出了一种移动峰函数（Moving Peaks Function），该函数可以描述为：函数曲线是 n 维的，包含有 m 个峰（Peaks）。环境每发生一次变化，各峰的高度、宽度和位置都会发生微小的变动，函数的适值定义为所有峰中的最大值。其表达式如下：

$$F(\boldsymbol{x},t)=\max(B(\boldsymbol{x}),\max_{i=1,\cdots,m}P(\boldsymbol{x},h_i(t),w_i(t),\boldsymbol{p}_i(t))) \tag{9.2}$$

其中，$B(\boldsymbol{x})$ 是一个非时变的“基本”曲线，P 是一个定义峰形的函数，每一个峰都具有自己的高度（h）、宽度（w）和位置（$\boldsymbol{p}$），这些参数也是时变的。

每经过 Δe 次估值，每个峰的高度、宽度和位置都会变化。每个峰的高度和宽度通过加上一个随机的高斯变量而变化，其位置的移动是由一个长度为 s 的向量 $\boldsymbol{v}$ 而引起的。这样参数 s（当然也包括高度和宽度变化的大小）可以用来控制一个变化的强度，Δe 能够决定环境变化的频率。一个新参数 $\boldsymbol{\lambda}$ 被用于决定一个峰位置的移动对它前一次移动的依赖程度。如果 $\boldsymbol{\lambda}=0$，则每次移动都是完全随机的，如果 $\boldsymbol{\lambda}=1$，则该峰总是向着一个方向移动（直到它撞到参数空间的边界，此时它将反弹回来）。

总之，参数 s 用于控制一个变化的强度，Δe 能够决定环境变化的频率，$\boldsymbol{\lambda}$ 被用于控制变化是否展示一种趋势。对于单峰的一个变化可由下列公式表示：

$$\sigma\in N(0,1) \tag{9.3}$$

$$h_i(t)=h_i(t-1)+height_severity\cdot\sigma \tag{9.4}$$

$$w_i(t)=w_i(t-1)+width_severity\cdot\sigma \tag{9.5}$$

$$\boldsymbol{p}_i(t)=\boldsymbol{p}_i(t)+\boldsymbol{v}_i(t) \tag{9.6}$$

这里的移动向量 $\boldsymbol{v}_i$ 定义为一个随机向量 $\boldsymbol{r}$ 和前一次移动向量 $\boldsymbol{v}_i(t-1)$ 的线性组合，并且将其长度标准化为 s，也就是

$$v_i(t)=\frac{s}{|r+v_i(t-1)|}[(1-\lambda)r+\lambda v_i(t-1)] \tag{9.7}$$

其中，随机向量 r 的产生是通过在每一维上产生一个随机数，再将其长度标准化为 s。

这类函数的复杂性可以很容易通过增加空间的维数和空间中峰的数量来实现，此外，还可以定义峰形函数 P 的不同表达形式和强度参数 s 的各种变化方法。

9.3.5 调度问题

现实世界中最常见的一类动态优化问题是调度问题（Scheduling Problems），特别是 Job Shop 调度问题。

通常调度问题的动态性主要体现在调度过程中随时会有新的工件到达。通常无论什么时候，一旦新工件到达，问题就会依据当前的最好解发生变化：所有在新工件到达前完成的操作被固定下来，并从优化问题中移出，此时的调度问题就由所有未完成的操作和新工件加工所需要的操作所组成。

很显然，对动态调度问题进行这样的处理会给算法带来两个挑战，第一，每次变化后基因编码方式会发生变化，也就是其长度必须增加以包含新到达的工件。第二，在某一特定时期内的最好解将会影响未来的适值曲线（因为每一次固定下来的操作依赖于当前的最好解），这会给算法间的比较带来更多的难度。

尽管变化频率可以通过一种直接的方式设定，然而变化的强度却很难控制，与大多数现实问题一样，进一步的分析更是难以进行。

Fang 等人提出了另一种动态的 Job Shop 调度问题，调度的工件数量被固定，而工件的加工时间和交货期会发生变化。虽然这种调度问题的研究不多，但是可以作为通向实际调度问题的一个中间阶段。

不幸的是，到目前为止看起来还没有任何形式的可公开使用的动态 JSSP 问题的标准测试数据，大多数调度问题的规模都相对较小。Bierwirth和 Mattfeld 在他们的研究中指出动态调度问题的规模应该足够大，以避免在仿真的开始或结束阶段产生某些无意识的影响。

为了构造一个简单却易于分析和进行参数调节的调度问题，Mattfeld，Engelmann 和 Branke 设计了一个带有交货期和安装时间的单机调度问题。因为这是一个单机问题，所以优化目标是获得工件在该机器上的最优加工序列，而任何一个序列都是可行的。此问题的动态性表现为两种模式：可变处理时间（Variable Processing Times）模式和滚动

时间窗（Rolling Time Horizon）模式。前者是指工件被固定，而处理时间和交货期会随时发生变化。在滚动时间窗模式中，环境每次变化时就会有 n 个新工件一批到达，此时在当前的最优解中所有在新工件到达前完成的工件要被固定下来，并从优化问题中移出。于是可以利用每次变化时到达的新工件数量 n 来控制变化强度，而变化间估值次数用以决定变化频率。

9.3.6 振荡峰函数

Branke 在检验记忆（Memory）对 EA 性能影响的实验中设计了一种振荡环境：在 n 维空间中，存在 l 条曲线（通常 $l=2$），每条曲线包含 m 个随机选择的峰（峰函数的定义见 9.3.4 节中的介绍），每条曲线按照一个 Cosine 函数振荡。这个振荡峰函数（Oscillating Peaks Function）的函数表达式为

$$f_i(t) = \omega(t) f_i(0) \tag{9.8}$$

$$w(t) = 0.5\cos\left(\frac{2t\pi}{steps} + 2\pi\frac{i-1}{l}\right) + 0.5, i = 1, \cdots, l \tag{9.9}$$

其中，$steps$ 表示一次循环中的中间步数。由此看来，函数实际上是在 l 条曲线之间振荡。

9.4 性能评估方法

在动态优化问题中通常并不存在唯一的非时变的最优解，因此算法的目标并不是为了找到某个极值点，而是尽可能紧密地追踪极值点在可行空间内的运动轨迹。为了比较不同算法之间性能的优劣，在当前的文献中研究者们常常使用一些能够显示算法聚集程度（也就是指每一代种群的最优适值和平均适值）的性能评估方法。

从目前的研究来看，下面这种数字评估方法可以作为用来比较动态环境中算法性能的不错选择。

在线（Online）性能和离线（Offline）性能最初是由 DeJong 提出的用于检验静态问题的性能评估方法，但很多研究者也将它们用在动态优化问题中。设：e_t 是第 t 次估值，T 是所考虑的估值次数。那么

（1）在线性能 x 能够通过计算整个循环中所有适值的平均值获得，即 $x = \frac{1}{T}\sum_{t=1}^{T} e_t$。由此可见，在线性能能够体现实际问题中每一次估值都需要检验的要求。

（2）离线性能 x^* 能够通过计算累积的当前平均的最优值获得，即 $x^* = \frac{1}{T}\sum_{t=1}^{T} e_t^*$，其中 $e_t^* = \max\{e_1, e_2, \cdots, e_t\}$。可见，离线性能是出于考虑在实际优化问题中仅仅需要考虑所获得的最优解。然而在非静态环境中，这种评估方法存在一定的问题，因为这里需要保证 e_t^* 是在当前环境中，算法到目前所能获得的最好解，而当环境发生一次变化后，所有个体需要被重新估值。因此，在计算动态环境中的离线性能时所考虑的个体应该是环境上一次变化后经过重新估值的个体，即 $x' = \frac{1}{T}\sum_{t=1}^{T} e_t'$ 其中，$e_t' = \max\{e_\tau, e_{\tau+1}, \cdots, e_t\}$，$\tau$ 表示环境最后一次变化时的估值次数。当然在使用离线性能时需要算法在其运行过程中能够探测到环境的变化。

如果实际的最优解可以随时获知，那么平均误差也可以作为一种性能评估方法，当然也有在线和离线两种模式，也要考虑环境变化时最优解和个体适值的变化情况。如果要防止出现较大的误差，则可以使用均方差指标。有时还可以使用相对误差（用最优解减去当前的最好解）来代替绝对误差来评估算法的性能。

在很多应用问题中，实际上关注的仅仅是当前所能获得的最好解，那么此时每种环境状态下所能获得的最好解的平均值也可以作为一种合适的评估指标。当算法不能连续地适应一个变化的环境，但还被要求在一定的预定时间内获得一个满意解时，这种性能的评估方法无疑是适用的。

从目前来看，离线性能和离线误差看上去是最合理的评估指标。这是由于随着算法的运行，这些累积的方法会使越来越多的估值进行平均。这样算法的性能曲线会变得越来越平滑，能够更准确地反映算法的长期性能。由于算法的追踪能力是动态优化问题中最令人感兴趣的性质，所以在评估算法性能的时候，有时候往往会忽略算法运行的最初几代（在此过程中，算法在寻找第一个最优解，可见这其实是一个静态阶段），而仅仅评估算法经过初始启动阶段之后的性能。如果算法在初始启动阶段的性能也是重要的，通常也将初始启动阶段的性能和长期运算性能分别进行比较。

9.5 探测环境中的变化

为了让算法能够直接对环境的变化做出反应，它首先要能够探测所

发生的变化。在很多研究中，变化对于系统来说是已知的，比如动态调度问题中要加入到调度序列中的已到达的新工件是已知的。然而，在另外一些情况下，例如原材料的质量可能随时发生变化，环境的变化是需要系统探测的。

如果能够假设环境的变化或多或少会引起当前解质量的下降，那么将监测整个种群性能或者其累积平均最优解的退化作为环境变化的探测器是一种最常用的方法。在一些研究中还发现，适值也可以被用来决定什么时候发现了新的最优解，也就是说，当旧的最优解被发现时也就决定了新的最优解至少在质量上要不弱于它。

Branke 采用了另一种略有不同的探测环境变化的方法，在算法运行的每一代，若干个个体都要被重新估值。如果其中至少有一个个体的适值发生了变化，那么就可以认为环境也发生了变化。

Fogarty 等人简要介绍了利用计算一个 EA 性能的确认模块来探测环境变化的方法，当某个限制值被超过时，EA 将被重启。也就是说，环境变化很大以至于算法的性能已经不再令人满意了。然而遗憾的是，作者并没有指出这个确认模块是怎样工作的。

还有一些探测环境变化的方法，比如通过建立当前环境的模型，并持续地监视这个模型是否与实际环境保持一致。如果利用模型预测的反应与从实际环境中获得的反应相差很大，那么就可以认为环境已经发生了变化。此时模型需要被更新，EA 要被重启。

9.6 原对偶遗传算法

近些年来，越来越多的研究者开始关注进化算法在动态优化问题中的应用，提出了很多改进策略并取得了很好的效果。本节接下来对 Yang 在 Proceeding of the 2003 Congress on Evolutionary Computation 提出的原对偶遗传算法（Primal - Dual Genetic Algorithm，PDGA）进行重点介绍，而对其他的适用于动态环境的进化算法仅作简单的介绍，如果读者对此感兴趣的话，请参考相关的综述性文献。

9.6.1 原对偶映射

通常动态环境需要 GA 保持足够的多样性才能持续地适应一个变化的适值曲线。为了改善 GA 在动态环境中的性能，一些研究者将自然界中普遍存在的补体和显性机制引入到算法中。在自然界和生物学中，互补现象是普遍存在的，例如

自然界

白天⇔黑夜

热带⇔极地

……

生物表现型

男　　性⇔女　　性

饱满种子⇔皱缩种子

……

生物基因型

DNA 的双链结构（A⇔T，G⇔C）

在双铰链的染色体中，由于显性机制的作用，使得染色体仅表达显性基因的性状，压制隐性基因的性状。

受这种补体和显性机制的启发，Yang 提出了用于动态 **0 - 1** 优化问题的 PDGA 算法。在 PDGA 中，每个染色体都在给定距离空间（比如 Hamming 距离空间）内定义了一个与之具有最大距离的对偶染色体。在 PDGA 运行过程中，在迭代进入下一代之前总是选择一部分低适值的个体进行对偶运算，并且给那些优秀的对偶染色体提供传递到下一代的机会。这种在原对偶染色体之间的原对偶映射（Primal - Dual Mapping，PDM）能够改善 PDGA 在搜索空间里的探索能力，使搜索更为有效率。

9.6.2 相关研究综述

在传统 EA 的运行过程中，随着迭代的进行，所有个体会逐渐收敛于某个局优点（或最优点）附近，这就使得种群失去其多样性，难以对环境的变化做出快速的反应。这就是 EA 在解决动态优化问题时所面临的主要挑战，近些年来，一些学者开始采用一些基于补体机制的方法来改善 EA 在动态环境中的性能，接下来本节将对这些方法进行简单的介绍。

最能体现基因补体机制的 GA 方法就是多倍体的方法。Goldberg 和 Smith 提出了基于二倍体和基因显性机制（Dominance）的 GA，由于无法预先确定特定基因是否为显性，他们采用了三等位基因的方案，也就是基因位值随机地选取为“0”，“隐性 1”或者“显性 1”，在动态背包问题的应用中，该方法比起简单 GA 来说表现出更好的性能。

但是 Goldberg 和 Smith 的三等位基因方法受到了 Ng 和 Wong 的质疑，他们认为三等位基因方法的结论是有异议的，所取得的结论之所以比单倍体染色体的 GA 好，主要是由于三等位基因 GA 的收敛速度较

慢，从而适应于论文中所使用的频繁变化的环境，而在其他的动态环境中，单倍体 GA 可能会取得比三等位基因 GA 更好的结论。Ng 和 Wong 提出了一种新的四等位基因的二倍体方法，为了更快地适应环境的变化，他们采用了一种基因显性变化机制，当一个个体的适值减少超过20%时，所有的等位基因位值都发生转换（即从显性转化成隐性，反之亦然）。他们在实验中发现这种二倍体方法要优于单倍体和三等位基因的二倍体方法。

Hadad 和 Eick 还提出了一种多倍体的方法，他们将一个显性变量作为个体的一部分，在动态背包问题中，这种方法能够获得与二倍体方法一样好的性能，而且发现多倍体的方法更适用于变化频率较高的环境。Ryan 使用了一种附加多倍体，附加的基因用于决定个体表现型的某个特性，当一个特定的阈值 b_1 被超过时，个体表现型的特性值变成1，反之为0。在后来的研究中，他对上述方法作了进一步的改进，当阈值 b_1 被超过时，个体表现型的特征为1；当这个值低于阈值 b_2 时，表现型的特征变为0；如果介于 b_1 和 b_2 之间，则表现型的特性随机选取。结果发现在某些动态问题中，这种方法要优于 Ng 和 Wong 的方法。

Lewis 等人在比较上述几种多倍体方法在振荡背包问题中的应用时发现仅采用简单的显性机制并不能保证能够有效地追踪到最优点，而采用了基因显性变化机制的二倍体方法，则取得了很好的效果。从目前的研究来看，多倍体的方法在周期性变化环境中表现出很好的性能，这主要是因为多倍体编码能够帮助 EA 记住环境所经历的变化状态，使得 EA 能够迅速地对环境的周期振荡做出反应。但是在非周期性变化环境或者经历的状态数较多的周期性变化环境中是否能够有效应用，依然存在着疑问。

Collard 等人提出的对偶 GA（Dual Genetic Algorithm，DGA）也是一种展现补体机制的方法。在 DGA 中，个体的基因型中被加入了一位元基因（位串的第一位），当元基因的位值为**0**时，没有什么影响，如果位值为**1**，则后面所有的位值将取反。这样搜索空间中每个点就有了两种互补的表现形式，比如说，个体［**0 011**］和［**1 100**］表示的都是空间中的同一个点，具有相同的表现型。在 DGA 中，元基因的使用会带来两个好处。首先，元基因的一个简单变异就会使个体完成其对补体的跳跃，也就是说，能够减小了搜索空间中两点间的最大 Hamming 距离。其次，即便当 GA 收敛于一个表现型时，个体的每一位基因都可能发生较大变化。因此，DGA 能够不依靠变异而是依靠“反射操作”就能使染色体发生很大变化，而对偶染色体之间的交叉运算可以看做是发

生了直接变异，因为两个染色体之间每一位都不相同。

9.6.3 PDGA 算法的框架结构

为了描述方便，首先介绍 PDGA 算法中的一些相关定义。

初始染色体（Primal Chromosome）：是指当前种群中直接记录的染色体。

对偶染色体（Dual Chromosome）：是指在给定距离空间中，与原染色体距离最远的染色体。记 $x' = dual(x)$，其中 x，x'分别表示原染色体和对偶染色体，$dual(\cdot)$ 表示原对偶映射（PDM）函数。

本节主要讨论二进制编码的 GA，通常使用 Hamming 距离作为 PD 映射函数。此时，两个染色体之间的 Hamming 距离指的是这两个染色体对应基因位点的值的不同个数。因此，在二进制空间中，一对染色体可以称为一对原对偶染色体，如果他们之间具有最大的 Hamming 距离，也就是说，若给定一个长度为 L 的染色体 $x = \{x_1, x_2, \cdots, x_L\} \in I = \{0, 1\}^L$，则它的对偶染色体（如图 9.1 所示）为 $x' = dual(x) = \bar{x} = \{\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_L\} \in I$，其中 $\bar{x}_i = 1 - x_i$（$i = 1, 2, \cdots, L$）。

原染色体:	0	1	0	1	1	0
PD映射:	↓	↓	↓	↓	↓	↓
对偶染色体:	1	0	1	0	0	1

图 9.1 一个 PD 映射的例子

优势染色体（Superior Chromosome）：是指一对原对偶染色体中适值较大的染色体，如果它们的适值不相等的话；而适值较小的染色体则被称为劣势染色体（Inferior Chromosome）。如果获得的对偶染色体是优势染色体，则一个 PD 映射能够被称为是有效的；否则这个 PD 映射就是一个无效的映射。PDGA 的基本思想就是希望通过有效的 PD 映射来改善 GA 的性能。

根据上面的定义，PDGA 算法的框架如图 9.2 所示，其中 N，P_c，P_m分别表示种群大小，交叉率和变异率，$f(x)$ 表示个体 x 的适值函数。在 PDGA 中，当产生一个中间种群 $P(t)$ 并完成常规的遗传运算（交叉变异运算）之后，再从 $P(t)$ 中选择一定数量 $D(t)$ 的个体，并计算它们的对偶染色体。对于任何 $x \in D(t)$，如果它的对偶染色体 x'适值更大，则 x 将被 x'所取代；否则 x 就保留下来。也就是说，只有有效的 PD 映射才能够使好的对偶染色体有机会传递到下一代种群中。

```
PDGA 的计算过程

开始
  参数设置（N，P_c，P_m）；
  令迭代指标 t=0；
  初始化种群 P(0)；
  对 P(0) 中的所有染色体进行估值；
  从 P(0) 中选择一部分个体构成 D(0)；
  for D(0) 中的每一个个体 x do
    计算 x 的对偶染色体 x'，并对 x'进行估值；//x' = dual(x)
    if f(x') > f(x)，then 用 x'替代 P(0) 中的 x；
  end for；
  repeat
  产生中间种群 P(t)；
  对 P(t)进行正常的遗传运算（交叉、变异等）；
  从 P(t)中选择一部分个体构成 D(t)；
  for D(t)中的每一个个体 x do
    计算 x 的对偶染色体 x'，并对 x'进行估值；//x' = dual(x)
    if f(x') > f(x)，then 用 x'替代 P(0) 中的 x；
  end for；
  t = t + 1；
  until 终止条件被满足；//例如，t > t_max
结束
```

图 9.2 PDGA 算法的简单流程

9.6.4 PDGA 中相关参数的讨论

1. 怎样构造 PDM 集 $D(t)$

从上面的讨论可以清楚地看到，从一个中间种群 $P(t)$ 中选择原染色体组成一个 PD 映射集 $D(t)$ 的目标应该是最大化有效的原对偶映射，也就是说，应该尽可能能从种群中选择劣势的原染色体。可见，选择方案的确定主要涉及 $P(t)$ 中什么样的原染色体应该被选择，数量是多少。

由于仅仅是有效的原对偶映射才能起作用，显然对低适值的染色体进行的原对偶映射更有可能是有效的。因此，总是从 $P(t)$ 中选择适值最小的个体来构成 $D(t)$。设 $n_d(t)$ 为第 t 代从 $P(t)$ 中选择原染色体组成 PDM 集 $D(t)$ 的大小，即 $n_d(t) = |D(t)|$。在算法运行的每一代，总是从 $P(t)$ 选择 $n_d(t)$ 个最小的原染色体用于对偶运算。现在剩下来

的问题是怎样确定 $n_d(t)$ 的值。

在文献中作者提出了一种适应性的方法来确定 $n_d(t)$ 的大小，设 $n_v(t-1)$ 是第 $t-1$ 代有效 PD 映射的实际数量，则 $n_d(t)$ 计算如下：

$$sign(t)=\begin{cases}1, if\dfrac{n_v(t-1)}{n_d(t-1)}<\delta \\ 0, if\dfrac{n_v(t-1)}{n_d(t-1)}=\delta \\ -1, if\dfrac{n_v(t-1)}{n_d(t-1)}>\delta\end{cases} \tag{9.10}$$

$$n_d(t)=\begin{cases}[\alpha\times N],\ t=0 \\ \min\{n_M,\max\{[\beta^{sign(t)}\times n_d(t-1),\ n_m]\}\},\ t>0\end{cases} \tag{9.11}$$

在式（9.10）中，$\delta\in(0,\ 1)$ 用来表示 $n_d(t)$ 变化的趋势：增大，减小或者保持不变；在式（9.11）中，$\alpha\in(0,\ 1)$ 和 $\beta\in(0,\ 1)$ 分别用于控制 $n_d(t)$ 的初始大小和增大或减小的速度，n_M，$n_m\in(0,\ N)$ 是预先设定的从 $P(t)$ 中选出的用于进行 PD 运算的原染色体最大和最小数量。在这种适应性的原染色体选择方案中，$n_d(t)$ 能够根据上一代原对偶映射中的有效映射所占的比率$\dfrac{n_v(t-1)}{n_d(t-1)}$是否超过某一给定的阈值 δ 进行适应性的调整。如果$\dfrac{n_v(t-1)}{n_d(t-1)}>\delta$，则 $n_d(t)$ 将增大；如果$\dfrac{n_v(t-1)}{n_d(t-1)}<\delta$，$n_d(t)$ 将减小；若相等，则 $n_d(t)$ 保持不变。这种选择方法会给 PDGA 带来更强的适应性，特别是在动态环境中，$n_v(t)$ 会随着时间的变化而变化。

2. 一种适应性的 PD 映射方法

上面讨论了在 PDGA 中如何适应性的调整 PD 映射集 $D(t)$ 的大小。值得注意的是，在 Yang 的 PDGA 中，PD 映射函数被设计为 Hamming 空间的最大距离，也就是说，在对一个原染色体进行 PD 运算时，它的每一位基因都要参加运算。然而很多时候，这种强烈的映射并不会改善算法的性能。比如在后期迭代过程中，种群中几乎所有的原染色体都分布在高性能（适值较大）的区域，此时它们的对偶染色体往往适值较低，这样几乎所有的 PD 映射都是无效的。而无效的 PD 映射除了浪费计算资源之外，既不能帮助种群获得更好的解，也不会丰富种群的多样性（后者对于解决动态优化问题更为重要）。另外，当环境发生一定的变化，但变化强度较小时，这种过于“强烈”的 PD 映射也无法明显改

善算法的性能。因此，为了改善PD映射的有效性，下面讨论一种新的适应性的基于统计的PD映射方法（Statistic - Based Primal-Dual Mapping，SPDM），考虑并不是让染色体的每一位都同时参加运算，而是根据当前种群的某种统计信息来决定它是否进行运算。

在这种基于统计的PD映射方法中，各基因位点的参与运算的概率是一个重要的变量，这里提出一种利用染色体各基因位点取值的统计信息来计算该位点参加PD运算概率的方法。设$p(i)$为基因位点i参加PD运算的概率，$i=1, 2, \cdots, L$；f_{si}为在种群中符号s在基因位点i上的出现频率，s为基因位点的编码取值。对于**0－1**编码空间，存在$f_{1i}+f_{0i}=1$，$0\leqslant f_{0i}$，$f_{1i}\leqslant 1$。f_{1i}能够被看做基因位点i上位值趋向于“1”的程度，如果所有f_{1i}都趋近于1或者0，那么整个种群就趋于收敛。可见由所有f_{1i}构成的一维向量$\{f_{11}, f_{12}, \cdots, f_{1L}\}$能够在基因层上反映这个种群的收敛程度。如图9.3所示，$p(i)$可以计算如下：

$$p(i)=\begin{cases}0.5-f_{1i}, & f_{1i}\leqslant 0.5\\ f_{1i}-0.5, & f_{1i}>0.5\end{cases} \tag{9.12}$$

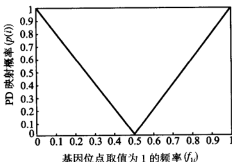


图9.3 基因位点参加PD映射的概率

很显然，对于二进制编码空间来说，利用f_{0i}来计算$p(i)$也会取得一样的效果。由图9.3中能够看出当$f_{1i}=0.5$时$p(i)=0$取得最小值；当$f_{1i}=1$或0时$p(i)=1$最大，也就是说在整个种群中某个基因位点的值越集中，该位点进行PD运算的概率越大。这主要是考虑到一个发散的种群能够更容易适应环境的变化。根据这种基于统计的映射方法，算法能够根据当前种群收敛程度对PD运算进行适应性调整。当$p(1)=p(2)=\cdots=p(L)=1$时，就是最初的PD映射方法。图9.4给出了一个对图中的染色体使用SPDM方法的例子。

1的频率：	0.1	0.4	0.8	0.3	0.7	0.2
映射概率：	0.8	0.2	0.6	0.4	0.4	0.6
是否运算：	1	0	0	0	1	1
原染色体：	0	1	0	1	1	0
PD映射：	↓				↓	↓
对偶染色体：	1	1	0	1	0	1

图 9.4 一个适应性映射方法的例子

9.6.5 PDGA 与 DGA

DGA 与 PDGA 的思想都是源于自然界中普遍存在的补体机制。DGA 是通过元基因的变异实现搜索空间中个体到其补体的跳跃；而 PDGA 中只有被选进映射集 $D(t)$ 中的原染色体才有机会发生这种跳跃。然而 PDGA 和 DGA 之间也存在许多不同，主要体现在 DGA 中互补染色体之间的跳跃是受变异驱策的，是随机发生的。它并没有采用显性机制，对补体机制的应用是很盲目的。而 PDGA 在染色体层采用了一种以适值作显性机制的方法，体现在它总是选择低适值的个体进行 PD 映射，而且仅当原染色体适值较差时才会被对偶染色体所替换。

在原始 PDGA 中，对偶染色体的表达是唯一的，而且仅仅被选入 PD 映射集中的染色体才有机会跳跃到其补体。但是由于在种群中仅仅记录原染色体，也就是说，染色体编码是一个单铰链而不是像 DNA 分子那样的双铰链结构，所以 Yang 的 PDGA 实际上是使用了一种伪二倍体的方法。而在采用 SPDM 方法的 PDGA（下文称之为 IPDGA）中，一个染色体补体的基因型具有多重的表达形式，这与自然界中存在的基因多倍性很相似。同样也是由于种群中仅仅记录原染色体，因此IPDGA可以被认为使用的是一种伪多倍体的方法。这样看来，IPDGA 能够保持更大的多样性，有助于更有效地探索解空间。此外，与原始 PDGA 相比，还有一点不同在于 IPDGA 要求每一个染色体都要参与 PD 运算，这也是出于进一步增强种群多样性的考虑。PDGA 与 IPDGA 的比较如图 9.5 所示。

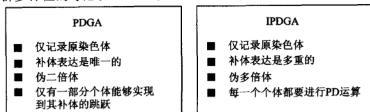


图 9.5 PDGA 与 IPDGA 的比较

9.6.6 PDGA的应用

接下来验证PDGA算法在两类动态优化问题中的性能。一类是简单的动态测试函数；另一类是在一种简单实际动态应用问题——动态背包问题。在实验中对IPDGA，PDGA，DGA和SGA进行比较，各种GAs都采用一样的遗传算子和参数：种群大小$N=100$；单节点交叉，交叉率$P_c=0.8$；位变异，变异率$P_m=0.001$；正比轮盘选择策略，并且总是将父代和子代的所有个体中最好的N个个体保留到下一代种群。对于PDGA，$\alpha=\beta=0.5$，$\delta=0.9$，$n_M=\dfrac{N}{2}$，$n_m=1$。所有的实验结果都是运行100次的平均值。

由于对于动态优化问题来说，并不存在单一的非时变的最优解，算法的目标并不是获得极值点，而是尽可能地跟踪它在解空间内的运动轨迹。于是采用一种离线性能指标——累积平均最优解x_T^*（也就是每一代获得的最优解的平均值）作为算法性能的评估指标。x_T^*可以计算如下：

$$x_T^* = \frac{1}{T}\sum_{t=1}^{T} e_t^* \tag{9.13}$$

其中，e_t^*是到t时刻为止所获得的最好解。可见随着迭代次数的增加，整个性能曲线会变得越来越光滑。

1. PDGA在简单动态测试函数中的仿真实验

在本实验中，动态测试函数是一些常用的标准静态优化问题通过一定的动态产生方法构造的。

（1）静态优化问题

① 位匹配问题

位匹配问题是一种常用于检验算法性能的One - Max问题，个体的适值等于与一个给定的字符串相匹配的位的总数量。这里使用了一个长度为100的二进制的位匹配函数，故其最优值为100。

② Royal Road函数

Royal Road函数最初是由Michel，Forrest和Holland提出的，这里定义在一个长度为64位包含有8个连续的8位Building Blocks的位串上。函数的适值能够计算如下：

$$f(x) = \sum_{i=1}^{8} c_i \cdot \delta_i(x) \tag{9.14}$$

其中，$c_i=8$，$\delta_i(x)=\begin{cases}\mathbf{1}, & x\in S\\ \mathbf{0}, & 其他\end{cases}$，$i=1, 2, \cdots, 8$，$S=\{s_1, s_2, s_3,$

s_4, s_5, s_6, s_7, s_8} 描述为

s_1 =11111111**

s_2 =********11111111**

s_3 =****************11111111**

s_4 =************************11111111********************************

s_5 =********************************11111111************************

s_6 =**11111111****************

s_7 =**11111111********

s_8 =**11111111

因此，函数的最优值是 64。

③ Deceptive 函数

Deceptive 函数是一族由低阶模式不能直接重组成高阶模式的函数，这里的 Deceptive 函数是由 10 个相同的 3 阶完全 Deceptive 函数 DF1 构造而成。DF1 被设计如下：

$$f(0\ \ 0\ \ 0)=28 \qquad f(0\ \ 0\ \ 1)=26$$
$$f(0\ \ 1\ \ 0)=22 \qquad f(0\ \ 1\ \ 1)=0$$
$$f(1\ \ 0\ \ 0)=14 \qquad f(1\ \ 0\ \ 1)=0$$
$$f(1\ \ 1\ \ 0)=0 \qquad f(1\ \ 1\ \ 1)=30$$

可见，函数的最大值为 300。

（2）动态测试环境的产生

下面介绍一种根据给定的静态函数产生动态优化问题的方法。对于一个给定的二进制编码静态优化问题 $f(x)$，构造一个动态环境的过程如下：首先，随机产生或按照某种控制方式产生一个二进制模板 T；然后对种群中的每一个染色体 x 执行位运算：$x\oplus T$，这里 $\oplus$ 是**异或**算子，例如 $\mathbf{1}\oplus\mathbf{1}=\mathbf{0}$，$\mathbf{1}\oplus\mathbf{0}=\mathbf{1}$，$\mathbf{0}\oplus\mathbf{0}=\mathbf{0}$。假设环境在第 t 代发生了变化，那么在第 $t+1$ 代，有 $f(x, t+1)=f(x\oplus T, t)$。

根据上述动态环境的产生方法可知，动态环境是通过对种群中所有个体的位运算而产生的，因此当环境变化后适值曲线依然会保持某些它的初始特性，比如最优解的数量和大小等。另外还能够发现，如果模板 T 中的某一位为 **1**，那么种群中染色体 x 在进行位运算时其对应位的值将会取反，也就是说，从 **0** 变为 **1** 或从 **1** 变为 **0**；如果 T 中的某一位为 **0**，则 x 的对应位将保持不变。可见模板 T 中 **1** 的位的总数能够表明此时环境变化的剧烈程度。用 s 表示模板 T 中 **1** 的位所占的比率，则可以用 s 作为环境变化的强度。当 s 等于 0.1，0.5 和 0.9 时分别可以表示微弱变化环境，随机变化环境和剧烈变化环境。

(3) 实验结果与分析

首先比较和分析了各种 GA 在动态位匹配问题中的实验结果。最大运行代数被设为 2000，适值函数曲线每 200 代发生一次变化，也就相当于各算法经历了 10 个环境的变化周期。图 9.6 显示了各种 GA 算法在不同变化强度的动态位匹配问题中的性能。

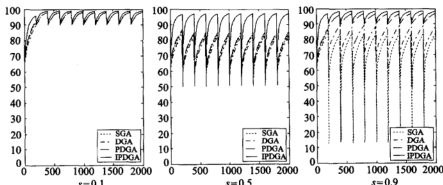


图 9.6 各种 GA 算法在不同强度的动态位匹配问题中的性能

从图 9.6 中能够发现，IPDGA 在各种环境特别是当 $s=0.5$ 时都要优于其他的 GA 方法。PDGA 的性能曲线在弱变化环境中与 DGA 和 SGA 的性能曲线交叠在一起，随着变化强度的增大，PDGA 的性能逐渐优于 DGA 和 SGA。DGA 的表现与 PDGA 类似，只不过在剧烈变化环境中其表现要远逊于 PDGA。

在 IPDGA 中，SPDM 算子有助于充分保持种群的多样性，当环境发生变化时，能够使个体快速地跳到最优点附近。而对于 PDGA，当环境变化不大时，大多数原对偶映射都是失效的，所以尽管原始的 PDM 算子也能帮助保持一定的多样性，但是其效率有时是很低的。由于元基因的变异是盲目的，所以与两种 PDGA 方法相比，DGA 在保持种群多样性方面表现更差。当环境发生变化时，SGA 在重新初始化时总是将旧种群中的部分个体（10%）保留下来，使得其性能曲线稍稍得到改善。

在 Royal Road 函数中采用了与位匹配问题一样的参数设置，实验结果如图 9.7 所示。从图 9.7 中能够看到，与位匹配问题相比，IPDGA的表现要远远优于 PDGA，DGA 和 SGA。而 PDGA 虽然总是能够比 DGA 和 SGA 获得更多的 BBs，但是却不能像 IPDGA 那样在各个动态周期内获得最优解。DGA 的表现令人失望，在某些时候甚至还

不如 SGA。

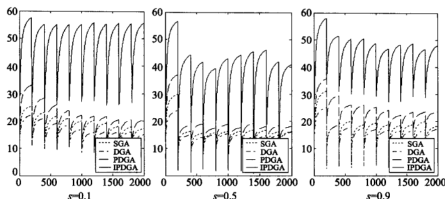


图 9.7 各种 GA 算法在不同强度的 Royal Road 函数中的性能

这是由于在 Royal Road 函数中一个基础 BBs 是 8 阶的，对于 GA 来说，想获得这样的 BBs 是比较困难的。IPDGA 中采用的适应性的补体机制能够给任何一个 block 直接转化为 BBs 的机会，而在 PDGA 中仅仅能够让所有位都是 0 的 block 直接转化为 BBs。可见，在这样的动态环境中 IPDGA 比 PDGA 表现得更为有效。

图 9.8 描述的是关于 Deceptive 函数的实验结果。与图 9.6 中的情况很相似，IPDGA 表现得最好，SGA 表现得最差，而 PDGA 的性能优于 DGA。

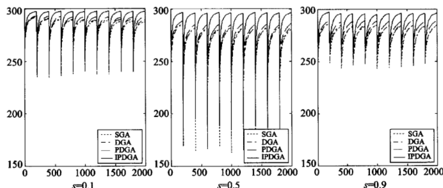


图 9.8 各种 GA 算法在不同强度的 Deceptive 函数中的性能

所有的实验结果都表明在动态环境中 IPDGA 比 PDGA 表现出更强的鲁棒性和适应性。而 DGA 采用的元基因方案虽然也能够改善算法在动态环境中的适应性，但是由于应用补体机制的盲目性，其效果受到了

限制。

2. PDGA 在动态背包问题中的应用

背包问题也是一类常用检验 GA 性能的组合优化问题，这类问题结构简单，既可以深入探索许多组合特性，又可以通过解决一系列背包问题来最终求解更为复杂的优化问题。

一个静态的背包问题可以描述为：假设需要从许多物品中选择一些来填充一个背包，存在 n 个不同的物品可以使用，每一个物品 j 具有重量 w_j 和价值 c_j。背包可以承重的上限为 W，问题是如何寻找物品的最优组合从而在满足背包承重约束的基础上，使装入背包中物品的总价值最大。费用、重量和承重一般都是正整数。

设

$$x_j = \begin{cases} 1, & \text{物品 } j \text{ 放入背包中}, \\ 0, & \text{其他} \end{cases} \quad j = 1, 2, \cdots, n \tag{9.15}$$

则 $x = (x_1, x_2, \cdots, x_n)$ 就能够表示放入背包中物品的一个组合，这样背包问题的数学描述如下：

$$\begin{aligned} \max f(x) &= \sum_{j=1}^{n} c_j x_j \\ \text{s. t.} \quad \sum_{j=1}^{n} w_j x_j &\leqslant W \end{aligned} \tag{9.16}$$

上述模型描述的背包问题就是所谓的 **0－1** 背包问题，是带有一个简单约束的纯整数规划，是一类非常重要的整数规划问题。为了便于 GA 求解，常常需要对模型中的约束条件进行处理。这里采用简单的惩罚系数法对模型公式（9.16）进行整理后，模型（9.16）转化为

$$\begin{aligned} \max f'(x) &= f(x) \cdot \left(1 - \frac{\left| \sum_{j=1}^{n} w_j x_j - W \right|}{W} \right) \\ &= \sum_{j=1}^{n} c_j x_j \cdot \left(1 - \frac{\left| \sum_{j=1}^{n} w_j x_j - W \right|}{W} \right) \end{aligned} \tag{9.17}$$

从式（9.17）中能够看出在适值计算过程中，不仅仅对不可行解（物品总重量超过背包的承重）进行惩罚，甚至也对某些可行解（物品总重量小于背包的承重）也进行一定的惩罚。

这里研究的是一个 50 个物品的动态 **0－1** 背包问题，其动态性主要体现在背包的承重会发生变化。在两种状态之间变化（背包的承重在

两个值之间变化）的背包问题可以看作是一种简单的振荡系统，这样系统的变化强度可以由两种承重量的差值大小所控制，物品的重量和价值见表 9.1。

表 9.1 50 个项目背包问题

项目号	重量 w_j	价值 c_j	项目号	重量 w_j	价值 c_j
1	80	220	26	28	90
2	82	208	27	30	88
3	85	198	28	22	82
4	70	192	29	50	80
5	72	180	30	30	77
6	70	180	31	45	75
7	66	165	32	30	73
8	50	162	33	60	72
9	55	160	34	50	70
10	25	158	35	20	69
11	50	155	36	65	66
12	55	130	37	20	65
13	40	125	38	25	63
14	48	122	39	30	60
15	50	120	40	10	58
16	32	118	41	20	56
17	22	115	42	25	50
18	60	110	43	15	30
19	30	105	44	10	20
20	32	101	45	10	15
21	40	100	46	10	10
22	38	100	47	4	8
23	35	98	48	4	5
24	32	96	49	2	3
25	25	95	50	1	1

在实验中，背包的承重分别在［796，1 249］和［658，1 334］之间进行变化。各GA算法的最大运行次数设为1 000次。每经过100代，背包的承重量发生一次变化（从一个值变为另一个值），也就是说整个环境将会振荡5次，其他相关参数设置与PDGA在简单动态测试函数中的仿真实验中的设置相同，实验结果如图9.9所示。

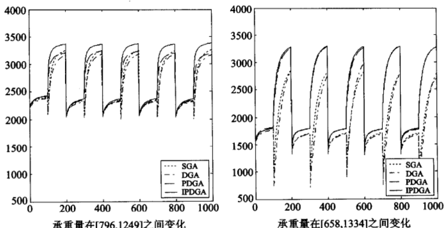


图9.9 各种GA算法在动态背包问题中的表现

从图9.9中能够看出，当环境振荡不是很强烈（承重量变化幅度较小）时，IPDGA表现出良好的性能；PDGA虽然表现也一直不错，但总是要逊于IPDGA；DGA的性能一直很差，而SGA却表现较好，甚至有的时候还要优于PDGA的性能。当环境振荡比较强烈（承重量变化幅度较大）时，IPDGA和PDGA的表现差不多，远远优于DGA和SGA；DGA的表现有所好转，有的时候要优于SGA。

这也是由于PDGA中的PD映射方法总是表现为基因型空间的最大距离，而IPDGA能够根据当前种群的收敛程度对PD映射方法进行适应性的调整，所以当环境变化较为强烈时，PDGA和IPDGA都能够通过有效的映射更快地找到较好的解，但是如果环境变化并不是很强烈时，IPDGA中的映射方法就比PDGA中的更为适用。而DGA由于其补体机制依赖于元基因位的随机变异上，具有一定的盲目性，所以它的表现并不理想。

受自然界中补体和显性机制的启发，一种PDGA算法被提出，并且对算法中的相关参数，比如PD映射集$D(t)$的大小、PD映射方法等进行了详细的讨论，通过实验证明该算法中的原对偶染色体的机制是一种有效的种群多样性保持的策略，特别是采用了SPDM方法后，PDGA在

某些动态环境中表现出很强的鲁棒性和适应性。

问题与思考

背包问题也可以采用自然数编码，试对文中的动态背包问题设计一种新的自然数编码空间内的 PDGA 算法，并比较与一般 GA 方法的性能，进一步将 PDGA 算法扩展到实编码空间。

参考文献

[1] Angeline P J. Tracking extrema in dynamic environments [C]. Proceedings of the Sixth International Conference on Evolutionary Programming, 1997, volume 1213 of LNCS.

[2] Bierwirth C, Kopfer H. Dynamic task scheduling with genetic algorithms in manufacturing systems [R]. Technical report, Department of Economics, University of Bremen, Germany, 1994.

[3] Bierwirth C, Kopfer H, Mattfeld D C, Rixen I. Genetic algorithm based scheduling in a dynamic manufacturing environment [C]. In Proceeding of IEEE conference on Evolutionary Computation, 1995.

[4] Bierwirth C, Mattfeld D C. Production scheduling and rescheduling with genetic algorithms [J]. Evolutionary Computation, 1999, 7(1): 1-18.

[5] Branke J. Evolutionary algorithms for dynamic optimization problems-a survey [R]. Technical Report 387, Insitute AIFB, University of Karlsruhe, February 1999.

[6] Branke J. Memory enhanced evolutionary algorithms for changing optimization problems [C]. In Proceeding of the Congress on Evolutionary Computation, 1999, 3: 1875-1882.

[7] Branke J, Mattfeld D, Engelmann T. A simple dynamic scheduling benchmark [M]. Evolutionary Optimization in Dynamic Environments Kluwer Academic Publishers, 2002, 24-25.

[8] Cobb H G. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environment [R]. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[9] Collard P, Escazut C, Gaspar A. An evolutionary approach for time dependant optimization [J]. International Journal on Artificial Intelligence Tools, 1997, 6 (4): 665 -695.

[10] Dasgupta D, McGregor D R. Nonstationary function optimization using the structured genetic algorithm [C]. In Proceeding of Parallel Problem Solving form Nature, 1992, 145 -154.

[11] De Jong K. An analysis of the behavior of a class of genetic adaptive systems [D]. PhD thesis, University of Michigan, Ann Arbor MI, 1975.

[12] Fang H L, R P, C D. A promising genetic algorithm approach to job - shop scheduling, rescheduling, and open - shop scheduling problems [C]. In Fifth International Conference on Genetic Algorithms, 1995, 375 -382.

[13] Fogarty T C, Vavak F, Cheng P. Use of the genetic algorithm for load balancing of sugar beet presses [C]. In International Conference on Genetic Algorithms, 1995, 617 -624.

[14] Fogel L J, Owens A J, Walsh M J. Artificial Intelligence through Simulated Evolution [M]. John Wiley, 1966.

[15] Ghosh A, Tstutsui S, Tanaka H. Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals [C]. In IEEE International Conference on Evolutionary Computation, 1998, 666 -671.

[16] Goldberg D E. Genetic Algorithms and Walsh Function: Part I, a Gentle Introduction. Complex System [M], 1989: 129 -152.

[17] Goldberg D E, Smith R E. Nonstationary function optimization using genetic algorithms with dominance and diploidy [C]. In Second International Conference on Genetic Algorithms, 1987, 59 -68.

[18] Grefenstette J J, Ramsey C L. An approach to anytime learning [J]. In Proceedings of the Ninth International conference on Machine Learning, 1992, 189 -195.

[19] Grefenstette J J. Genetic algorithms for changing environments [J]. In Proceeding of Parallel Problem Solving from Nature 2, 1992, 137 -144.

[20] Hadad B S, Eick C F. Supporting polyploidy in genetic algorithms using dominance vectors [J]. In Proceedings of the Sixth Internation-

al Conference on Evolutionary Programming, 1997, 223 -234.

[21] Karr C L. Genetic algorithms and fuzzy logic for adaptive process control [J]. In Intelligent Hybrid Systems, 1995, 4: 63 -83.

[22] Karr C L. Adaptive process control using biologic paradigms [C]. Proceedings of Electronic Technology Directions to the Year 2000, 1995, 1: 128 -136.

[23] Lewis J, Hart E, Ritchie G. A comparison of dominance mechanisms and simple mutation on non-stationary problems [C]. In Proceeding of Parallel Problem Solving from Nature, 1998, 139 -148.

[24] Mitchell M, Forest S, Holland J H. The Royal Road for Genetic Algorithms: Fitness Landscape and GA performance [C]. In Proceeding of 1st European Conference on Artificial Life, 1992: 245 - 254.

[25] Mori N, Imanishi S, HKita, Nishikawa Y. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm [C]. In Seventh International Conference on Genetic Algorithms, 1997, 299 -306.

[26] Mori N, Kita H, Nishikawa Y. Adaptation to a changing environment by means of the thermodynamical genetic algorithm [C]. In Proceeding of Parallel Problem Solving from Nature, 1996, 513 -522.

[27] Mori N, Kita H, Nishikawa Y. Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm [J]. In Proceeding of Parallel Problem Solving from Nature, 1998, 149 -158

[28] Ng K P, Wong K C. A new diploid scheme and dominance change mechanism for non - stationary function optimization [C]. In Sixth International Conference on Genetic Algorithms, 1995, 159 -166.

[29] Ryan C. Diploidy without dominance [C]. In Third Nordic Workshop on Genetic Algorithms, 1997, 63 -70.

[30] Ryan C, Collins J J. Polygenic inheritance-a haploid scheme that can outperform diploidy [C]. In Proceeding of Parallel Problem Solving from Nature, 1998, 178 -187.

[31] Smith R E. Diploid genetic algorithms for search in time varying environments [J]. In Annual Southeast Regional Conference of the ACM, 1987, 175 -179.

[32] Vavak F, Fogarty T C, Jukes K. A genetic algorithm with variable

range of local search for tracking changing environments [J]. In Proceeding of Parallel Problem Solving from Nature, LNCS 1141, 1996.

[33] Vavak F, Fogarty T C, Jukes K. Learning the local search range for genetic control of dynamic systems [C]. Proceedings of the pre - conference workshop on Evolutionary computing and machine Learning, 1996, LCML96: 143 - 150.

[34] Vavak F, Fogarty T C, Jukes K. Adaptive balancing of a bank of sugar - beet presses using a genetic algorithm with variable local search range [C]. In 3rd Intl. Mendel Conference on Genetic Algorithms, 1997, 164 - 169.

[35] Vavak F, Fogarty T C, Jukes K. Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search [C]. Seventh International Conference on Genetic Algorithms. Morgan Kaufmann, 1997, 719 - 726.

[36] Wang Hongfeng, Wang Dingwei. An Improved Primal - Dual Genetic Algorithm for Optimization in Dynamic Environments [J]. 13th International Conference on Neural Information Processing, LNCS 4234, 2006, 836 - 844.

[37] Yang S. Non - stationary problem optimization using the primal - dual genetic algorithm [J]. Proc. of the 2003 Congress on Evolutionary Computation, 2003, 4: 2246 - 2253.

[38] 王洪峰，汪定伟．动态环境中的进化算法 [J]．控制与决策，2007，22 (2)：127 - 131.

结 束 语

优化是从函数优化发展起来的。第二次世界大战之后，随着运筹学的产生与发展，线性规划、非线性规划、动态规划等数学规划的方法获得了长足的发展和广泛的应用。但是传统的数学规划方法对问题的描述有严格的要求，一般要求问题的目标函数和约束必须是线性的，至少是连续可微的，并要求目标函数是凸函数，约束集是凸集才能保证解的最优性。这些严格的限制大大缩小了数学规划方法的应用范围。

自20世纪70年代末起，以遗传算法、禁忌搜索和模拟退火为代表的智能优化算法迅速发展起来，这些方法有广泛的普适性，对目标函数和约束集没有严格要求，在许多行业里都获得了广泛的应用。以这些方法为主干的软计算、进化计算，或称为高级启发式，已成为最优化方法应用的主流。这本教材介绍的就是这一阶段的优化方法。

但是，随着信息技术、先进制造技术的发展，人们面临的系统越来越大，越来越复杂，各种新的资源优化配置问题对最优化方法提出了一些新的要求。比如非集中式、非时序性、非平衡性等，现有的智能优化的方法也不能满足这些要求。

随着实际需要的不断提高，最优化的发展又出现了一些新的趋势。

(1) 从方法定向到应用定向转变——最优化已从问题适应方法转变为方法适应问题。

(2) 问题的信息特征越来越复杂——非线性、非连续、非对称、非一致、可容错都成为必须面对的问题。

(3) 从集中式到分布式转变——无论数学规划还是进化计算都是集中式算法，即有一个唯一的权威决策者；多决策者自组织、自协调、互相替补的优化方法已经呼之欲出。

(4) 从因果性到非因果性（Non - Causality）转变——迄今为止谈论的问题都是因果的，正常时序的，即前因后果。由于复杂社会和物理系统中非时序现象的存在，这类系统的优化就要求非因果性的方法。

(5) 从平衡态向非平衡态转变——现有优化方法基本上都是针对处于平衡态的问题的，近年来突变性（Catastrophe）和突发性（Emergency）已越来越引起全社会的关注，优化方法势必也必须面对这类问题。

近年来，出现了一批模仿生命系统特性的计算方法，人们熟知的人

工神经网络、遗传算法、人工免疫系统、蚁群算法、进化计算便是这类方法的典型代表。实际上，类似的计算方法仍在不断地出现，比如，模拟食肉动物捕食方法的掠夺搜索策略（Predatory Search Strategy），模拟生物食物链的人工生命算法（Artificial Life Algorithm）。可以预期，随着生命科学和信息科学的迅猛发展和不断地交叉融合，这类模拟生命系统的计算方法将会不断涌现。同时，从现阶段研究情况看这类模拟生命系统的计算方法还是点对点的模仿，对生命系统机理的理解和借鉴的层次还比较低，多数还停留在简单模仿阶段。

自 1987 年 Langton 提出人工生命的概念后，人工生命的研究和应用获得了长足的发展。从最小的细胞级到器官级、个体级、种群级，乃至最大的含多物种的生态系统级的人工生命系统都在研究之中。大自然中有着亿万种不同的物种，其行为、功能和习性千奇百怪、精彩纷呈，为人工生命系统的学习和模仿提供了大量的样本。这就启发人们思考，用人工生命的方法模仿生命系统的多种功能习性，创造出一系列全新的复杂系统的计算方法。这类方法可以统称为人工生命计算（Artificial - Life Computation）。

人工生命计算是借鉴模拟生命系统功能、行为和特性的科学计算方法；是生命科学、信息科学和运筹学的交叉研究领域；是进化计算的一个新的分支；是由具有生命特性的多智能体以特定计算目标为依据，有序组合起来所形成的计算方法。

人工生命计算的主要研究内容包括：

（1）对现有的模拟生命系统的算法，包括人工神经网络、遗传算法、人工免疫系统、蚁群算法、进化计算、人工生命算法等进行综述、分析和分类。

（2）基于上述分析，归纳出在不同算法中模拟生命系统的规律、法则和方法论，总结出相应的理论。并进一步提出人工生命计算的理论框架。

（3）学习生命科学中的理论，包括遗传学、免疫学、分子生物学、生态学，动物学和植物学，根据第二项中的理论分析哪些生命的行为、习性和机制具有供科学计算借鉴和模仿的潜力。

（4）根据以上分析，设计开发针对不同应用的全新的人工生命计算方法。

自然界生命存在的形式丰富多彩；不同生命的功能和习性千变万化；这些功能习性是在大自然中经千百万年的考验而形成的，是宝贵的自然财富。设计新算法的准则：先学习再模仿最后改进；针对一定的目

标需要；抽取信息特征，改变物质能量的依附关系；这样，可以创造出的新方法应该是无穷无尽的。

近年来国内一些学者已经提出了一些具有人工生命计算性质的算法。如：鱼群算法、食物链算法、雁队算法、群落选址算法等。但是由于算法的性能还不能和已有的算法媲美，要得到国内外同行的普遍公认还需要一段过程。

我们希望对新的优化算法有兴趣的读者，大胆创新、严谨验证，投入到这个领域中来，开发出适应新的实际需要的优化算法，为开创新的研究领域作出贡献。

参考文献

[1] Hayashi D et al. Distributed optimization by using artificial life [J]. Tras. IEE Japan, 1996, 116 - C (5): 584 - 590.

[2] Langton C G. (edited) Artificial Life [M]. Addison - Wesley Publishing Company, 1989.

[3] Linhares A. Synthesizing a Predatory Search Strategy for VLSI Layouts [J]. IEEE Trans. On Evolutionary Computation, 1999, 3 (2): 147 - 152.

[4] Wang D. Colony location algorithm for assignment problems [J]. Journal of Control Theory and Applications, 2004, 2 (2): 111 - 116

[5] 王寿云，于景元，戴汝为．开放的复杂巨系统 [M]．杭州：浙江科技出版社，1996.

[6] 喻海飞，汪定伟．基于人工生命的食物链算法及其在供应链计划中的应用 [J]．系统仿真学报，2005，17 (5)：1195 - 1199.

[7] 张永光．“人工生命”研究进展 [J]．中国科学院院刊，2000 (3)：169 - 173.